Software Qual J (2016) 24:755–795
DOI 10.1007/s11219-015-9291-5

A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems

Moharram Challenger[1] · Geylani Kardas[1] · Bedir Tekinerdogan[2]

Published online: 10 September 2015

Abstract Multi-agent systems (MASs) include multiple interacting agents within an environment to provide a solution for complex systems that cannot be easily solved with individual agents or monolithic systems. However, the development of MASs is not trivial due to the various agent properties such as autonomy, responsiveness, and proactiveness, and the need for realization of the many different agent interactions. To support the development of MASs various domain-specific modeling languages (DSMLs) have been introduced that provide a declarative approach for modeling and supporting the generation of agent-based systems. To be effective, the proposed DSMLs need to meet the various stakeholder concerns and the related quality criteria for the corresponding MASs. Unfortunately, very often the evaluation of the DSML is completely missing or has been carried out in idiosyncratic approach. If the DSMLs are not well defined, then implicitly this will have an impact on the quality of the MASs. In this paper, we present an evaluation framework and systematic approach for assessing existing or newly defined DSMLs for MASs. The evaluation is specific for MAS DSMLs and targets both the language and the corresponding tools. To illustrate the evaluation approach, we first present SEA_ML, which is a model-driven MAS DSML for supporting the modeling and generation of agent-based systems. The evaluation of SEA_ML is based on a multi-case study research approach and provides both qualitative evaluation and quantitative analysis. We report on the lessons learned considering the adoption of the evaluation approach as well as the SEA_ML for supporting the generation of agent-based systems.

✉ Geylani Kardas
geylani.kardas@ege.edu.tr

Moharram Challenger
moharram.challenger@mail.ege.edu.tr

Bedir Tekinerdogan
bedir.tekinerdogan@wur.nl

[1] International Computer Institute, Ege University, Izmir, Turkey

[2] Information Technology Group, Wageningen University, Wageningen, The Netherlands

Keywords Domain-specific modeling language (DSML) · Evaluation framework · Evaluation of DSMLs · Multi-agent system (MAS) · Software agent · SEA_ML

1 Introduction

Software agents are considered as autonomous software components which are capable of acting on behalf of their human users in order to perform a group of defined tasks (Wooldridge and Jennings 1995). In its most fundamental artificial intelligence definition, an agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors (Russell and Norvig 2003). Multi-agent systems (MASs) include multiple interacting agents within an environment to provide a solution for complex systems that cannot be easily solved with individual agents or monolithic systems.

However, the development of MASs is not trivial due to the various agent properties such as autonomy, responsiveness, and proactiveness, and the need for realization of the many different agent interactions. The interaction among agents can be based on various complex protocols and can be either cooperative or selfish (Sycara 1998).

Therefore, it is natural that methodologies are being applied to master the problem of defining such complex systems. One of the possible alternatives is represented by domain-specific languages (DSLs) (van Deursen et al. 2000; Mernik et al. 2005; Varanda-Pereira et al. 2008; Fowler 2011) that have notations and constructs tailored toward a particular application domain (e.g., MAS). The end users of DSLs have knowledge from the observed problem domain (Sprinkle et al. 2009), but they usually have little programing experience. Domain-specific modeling languages (DSMLs) (Mernik et al. 2005; Kelly and Tolvanen 2008; Fister et al. 2011) further raise the abstraction level, expressiveness, and ease of use. The main artifacts of DSML are models instead of software codes, and they are usually specified in a visual manner (Schmidt 2006; Gray et al. 2007).

To support the development of MASs, various DSMLs have been introduced that provide a declarative approach for modeling and supporting the generation of agent-based systems (e.g., Rougemaille et al. 2007; Hahn 2008; Fuentes-Fernandez et al. 2010; Ciobanu and Juravle 2012; Gascuena et al. 2012; Challenger et al. 2014). In addition to these existing DSMLs, it is expected that new DSMLs can be introduced in the future to address various different concerns in MAS. DSML allows end-user programmers and domain experts to specify the core essence of a problem using visual abstractions that are close to the problem space of a specific domain.

To be effective, the proposed DSMLs need to meet the various stakeholder concerns and the related quality criteria for the corresponding MASs. Several studies have reported on the successful application of DSMLs in different companies such as SAP, Telephonica and WesternGeco (Mohagheghi et al. 2013). Unfortunately, very often the systematic evaluation of the DSML is completely missing or has been conducted in an idiosyncratic way (Mohagheghi and Dehlen 2008). That is because DSMLs have a relatively short history, and research on DSML evaluation methodologies is currently in its preliminary stage where researchers still continue to work especially on the derivation of appropriate DSML evaluation criteria (Kahlaoui et al. 2008; Barisic et al. 2012a). Moreover, experience reports with comprehensive evaluations and the adopted criteria may be kept private and not published (Mohagheghi and Dehlen 2008) because of their business value. Hence, public reports on the real and complete evaluation of DSMLs in various fields can be beneficial especially for researchers and developers who intend to use a DSML-based engineering approach.

If MAS DSMLs are not well defined, then implicitly this will have an impact on the quality of the MASs. Several criteria need to be taken into account that will impact the effectiveness of the MAS DSML. In this context, we present an evaluation framework and systematic approach for the assessment of existing or newly defined DSMLs for MASs including both the language and supporting tools of the language. The framework mainly addresses the concerns of MAS stakeholders for both qualitative evaluation and quantitative analysis, so targeted languages are DSMLs specific for MAS rather than DSMLs in general. The qualitative evaluation includes criteria for supporting the general DSML assessment, and for the end user's perspective. The quantitative analysis focuses on the executional dimensions. This evaluation also paves the way for comparison and selection of MAS DSMLs.

Although the assessment criteria presented in the evaluation framework take into account the evaluation of MAS DSMLs and not their development processes, the evaluation framework can also be used to evaluate parts of a MAS DSML during its development (and/or compare with another mature MAS DSML), e.g., its abstract syntax, concrete syntax, and transformations.

To illustrate the evaluation approach we first present SEA_ML, which is a MAS DSML for modeling and generation of agent-based systems (Challenger et al. 2014). SEA_ML is a comprehensive DSML that realizes the concerns of many existing DSMLs to especially support the development of software agents in the Semantic Web environment (Berners-Lee et al. 2001; Shadbolt et al. 2006). As such, the interactions and applied protocols between agents and semantic web services are covered by SEA_ML (Getir et al. 2012). The evaluation of SEA_ML is based on a multi-case study research approach. For evaluating the language elements of SEA_ML, the specifications of three main parts of this DSML including the abstract syntax, concrete syntax and code generation are analyzed. The evaluation considers the application of SEA_ML to develop agent-based systems. However, SEA_ML is used here for the demonstration purposes, and the proposed evaluation framework can already be employed for the assessment of any MAS DSML due to its built-in specifications and defined criteria by taking into consideration all aspects of MAS modeling at static and dynamic levels; including both agent internals and MAS environmental and organizational structures.

For evaluating the executional dimension, the generated artifacts are compared with the final artifacts, and the percentages of generated artifacts are calculated. In order to assess the development using SEA_ML, we have adopted two groups of users who had to develop MASs for four different case studies. The first group used SEA_ML, while the members of the second group did not use SEA_ML and they were free to use general-purpose modeling languages. We call this group as manual developers. Manual users were also free to implement systems by using general-purpose programing languages without a preceding modeling stage. The comparison between MAS DSML-based development and manual MAS development [e.g., by using a general-purpose language (GPL)] can be used to evaluate DSMLs. That approach is similar to the comparison between a GPL and a DSL given in Kosar et al. (2010). DSML-based development aims to be substituted with manual development where DSML's performance takeovers the manual approach. We report on the results of the case study research and provide the lessons learned considering the adoption of the evaluation approach as well as the development of SEA_ML for supporting the generation of agent-based systems.

The remainder of the paper is organized as follows: Sect. 2 discusses the background of the study, including a general description of DSMLs for MAS and an example DSML, SEA_ML. Section 3 presents the evaluation framework. Section 4 describes the multi-case study protocol in which we adopt the evaluation framework to illustrate the evaluation of SEA_ML. Section 5 presents the execution of the case studies including the data extraction. Section 6 presents the results and their analysis for the case study research. Section 7 includes the discussion on threats to validity. Section 8 describes the related work, and finally we conclude the paper with Sect. 9.

2 DSML for MASs

In this section we present the basic elements of a MAS DSML and describe SEA_ML which is an example MAS DSML that will be used throughout the paper to illustrate the evaluation approach. The general language elements for MAS DSMLs are presented in Sect. 2.1, while SEA_ML is presented in Sect. 2.2.

2.1 Elements of MAS DSMLs

In general, a DSML is defined using an abstract syntax, concrete syntax, static semantics, and the operational semantics of the language (Warmer and Kleppe 1998; Mernik and Zumer 2005). The abstract syntax of a DSML describes the concepts and their relations to other concepts without any consideration of representation or meaning. In other words, the abstract syntax of a language describes the vocabulary of concepts provided by the language and how they may be combined to form models or programs (Clark et al. 2004). In terms of model-driven development (MDD), the abstract syntax is described by a metamodel which defines what the models should look like. The concrete syntax definition provides a mapping between meta-elements and their representations. Static semantics provides definitions of additional constraint rules on abstract syntax that are hard or impossible to express in standard syntactic formalisms of the abstract syntax (Saritas and Kardas 2014). Finally, the operational semantics of the language provides the meaning of the concepts defined in the abstract syntax.

As stated before, various MAS DSMLs have been introduced in the literature. These DSMLs are based on well-defined metamodels including several viewpoints (Demirli and Tekinerdogan 2011) to address the domain concepts (e.g., Bernon et al. 2005; Omicini et al. 2008; Beydoun et al. 2009; Hahn et al. 2009; Kardas et al. 2009; Challenger et al. 2011).

In Fig. 1, for the demonstration purposes, we illustrated a simplified and generic MAS metamodel that may be considered as a high-level abstraction. It is worth noting that an actual MAS metamodel is much more complicated with including many entities and their associations as can be seen in various widely known agent metamodels (e.g., Omicini et al. 2008; Beydoun et al. 2009; Hahn et al. 2009; Kardas et al. 2009; Challenger et al. 2014). Further, actual MAS metamodels can consist of different viewpoints such as agent internal, environment, interaction/collaboration, protocol and organization. Simplified metamodel given in Fig. 1 is used to describe and explain below some of the important concepts and language elements of MAS DSMLs.

According to the metamodel in Fig. 1, a MAS consists of organizations in which agents work. Each agent has its goals, plans, and beliefs which constitute the agent's capacities.

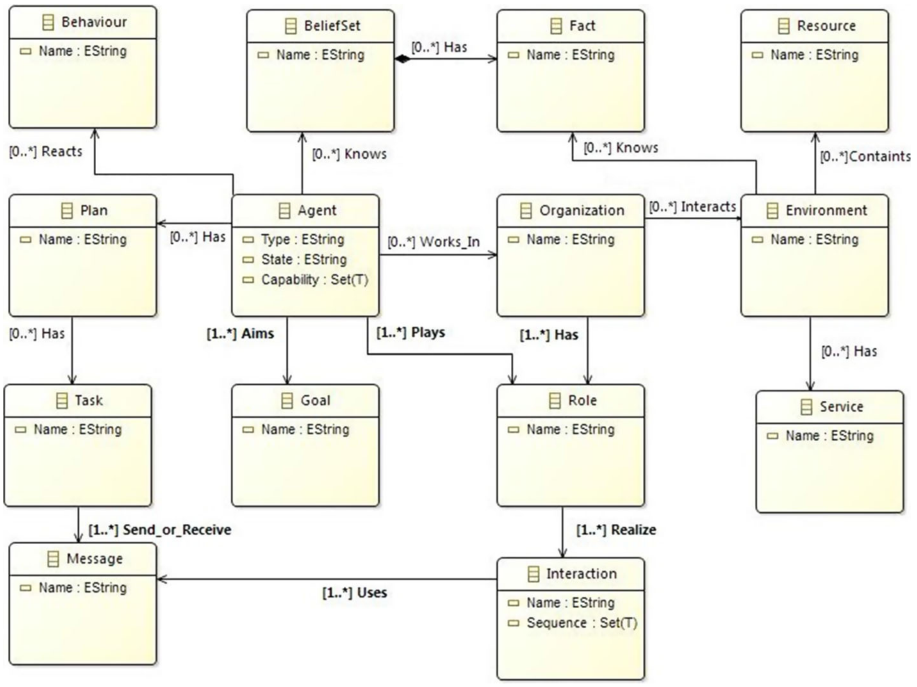

Fig. 1 General metamodel (abstract syntax) representing MAS DSMLs

These components are required especially for Belief–Desire–Intention (BDI) agents (Rao and Georgeff 1995) to be able to realize the pro-active feature of agents. On the other hand, an agent having its behaviors can react to the events and realize reactiveness of agents. Agents play roles to reach their goals, and they can change these roles during time and/or moving from organization to another one. Inside these roles, agents may also interact by message passing. These messages are sent and received, while agents execute their tasks which compose plans in the agents. Finally, agents can change their organization and use resources, information, and services which are in the environments of those organizations.

2.2 SEA_ML

In this paper, we will use SEA_ML (Challenger et al. 2014) as the instance DSML to exhibit how the proposed evaluation approach can be employed. SEA_ML is a MAS DSML for the development of Semantic Web enabled MASs. Main goal of this language is to provide a convenient and handy environment for agent developers to construct and implement software agent systems working on various application domains. In order to support MAS experts when programing their own systems, and to be able to fine-tune them visually, SEA_ML covers all aspects of an agent system from the internal view of a single agent to the complex MAS organization. In addition to these capabilities, SEA_ML also supports the model-driven design and implementation of autonomous agents who can evaluate semantic data and collaborate with semantically defined entities of the Semantic Web, like Semantic Web Services (SWS). That feature exactly differentiates SEA_ML and makes unique regarding any other MAS DSML currently available. Within this context, it includes new viewpoints which specifically pave the way for the development of software agents working on the Semantic Web environment. Modeling agents, agent knowledge-bases, platform ontologies, semantic web services and interactions between agents and SWS are all possible in SEA_ML. Taking into account all of these features, this full-fledged MAS DSML is chosen for the study reported in this paper.

SEA_ML's metamodel is divided into eight viewpoints, each of which represents a different aspect for developing Semantic Web enabled MASs. These viewpoints include the followings: Agent's Internal Viewpoint is related to the internal structures of semantic web agents (SWAs) and defines entities and their relations required for the construction of agents. It covers both reactive and Belief–Desire–Intention (BDI) (Rao and Georgeff 1995) agent architectures. Interaction Viewpoint expresses the interactions and communications in a MAS by taking messages and message sequences into account. MAS Viewpoint solely deals with the construction of a MAS as a whole. It includes the main blocks which compose the complex system as an organization. Role Viewpoint delves into the complex controlling structure of the agents and addresses role types. Environmental Viewpoint addresses the use of resources and interaction between agents with their surroundings. Plan Viewpoint deals with an agent's Plan's internal structure, which are composed of Tasks and atomic elements such as Actions. Ontology Viewpoint addresses the ontological concepts which constitute agent's knowledgebase (such as belief and fact). Agent–SWS Interaction Viewpoint defines the interaction of agents with semantic web services (SWS) (Sycara et al. 2003) including the definition of entities and relations for service discovery, agreement and execution.

The collection of SEA_ML viewpoints constitutes an extensive and all-embracing model of the MAS domain. The SEA_ML views, namely MAS, Agent Internal and Interaction, cover the main agent entities and their relations on which the agent research community is mostly agreed. Similar abstractions can also be found in (e.g., Pavon et al. 2006; Omicini et al. 2008; Hahn et al. 2009; Beydoun et al. 2009). Furthermore, more specific aspects of the domain like Plan, Role and Environment are also supported in SEA_ML syntax with individual metamodels. In fact, proposing specific metamodels for these viewpoints is not a new idea. For example, Hahn (Hahn 2008) also presents sub-metamodels for the Plan and Role aspects, while the Environment view is specifically considered in Omicini et al. (2008) and Challenger et al. (2011). However, SEA_ML's abstract syntax combines all of these generally accepted aspects of MAS and introduces two new viewpoints (Agent–SWS Interaction and Ontology) for supporting the development of software agents working within the Semantic Web environment. On the other hand, the behavior and goal aspects can also be considered with distinct sub-metamodels although such an approach is not very common in MAS DSML proposals. Due to the generality considerations and the scarcity of meta-entities for being a unique model, SEA_ML does not consider these aspects as individual sub-metamodels. Instead, the required first class entities and their relations pertaining to the behavior and goal aspects of agent systems are included within the agent internal viewpoint of SEA_ML.

SEA_ML can be both used for modeling MASs and generation of code from the defined models. SEA_ML instances are given a series of model-to-model (M2M) and model-to-text (M2T) transformations to achieve executable artifacts of the system-to-be-built for various agent platforms [such as JACK (AOS 2001) and JADEX (Pokahr et al. 2005)] and semantic web service ontologies [e.g., OWL-S (Martin et al. 2004)]. For implementing the M2M transformations, ATL language (Jouault et al. 2008) was used, while the M2T transformations were implemented using MOFScript (Oldevik et al. 2005; MOFScript 2005). It is also possible to automatically check the integrity and validity of SEA_ML models (Getir et al. 2014). Complete discussion on SEA_ML and supported development mechanism with including the internals of the abovementioned transformations can be found in Challenger et al. (2014).

Comparison results again given in Challenger et al. (2014) show that SEA_ML and the studies in Hahn (2008), Hahn et al. (2008), Fuentes-Fernandez et al. (2010), Ciobanu and Juravle (2012) and Gascuena et al. (2012) have the expected features of being an almost complete DSL/DSML for the development of agent systems since those studies cover the definition of an abstract syntax, a concrete syntax and operational semantics at least, and provide some mechanisms for the exact implementation of MASs. However, some of them do not consider the general purpose. For instance, DSL introduced in Ciobanu and Juravle (2012) only provides for the development of mobile agents, while Gascuena et al. (2012) introduce a model just within the scope of Prometheus methodology. On the other hand, SEA_ML and DSML4MAS (Hahn 2008) differentiate from the remaining studies by introducing general metamodels with various viewpoints that enable the development of MAS for many application domains. Further, SEA_ML supports the interaction of agents with semantic web services and introduces new viewpoints for agent internals when considering semantic web support. Although it can be said that Hahn et al. (2008) also share the same service interaction perspective, SEA_ML brings a different approach by the provision of a unique metamodel. Moreover, as far as we know, SEA_ML is also the first MAS DSL which provides both textual (Demirkol et al. 2013) and graphical concrete syntaxes for MAS developers.

Finally, it is worth indicating that abovementioned MAS DSML studies bring noteworthy contributions on MDD of MASs by especially focusing on agent metamodeling, model transformation and tooling aspects. However, they do not involve any comprehensive evaluation of the proposed languages including quantitative and/or qualitative assessment of language features, development environment, and etc. based on an empirical work.

3 Evaluation framework

In this section, we present the framework for evaluating MAS DSMLs. The framework covers the evaluation of MAS DSML elements, tooling and generated artifacts. Figure 2 provides a conceptual model which describes the overall context in which the evaluation framework is used. In the figure, *Evaluation Approach* presented in this study takes a *MAS*

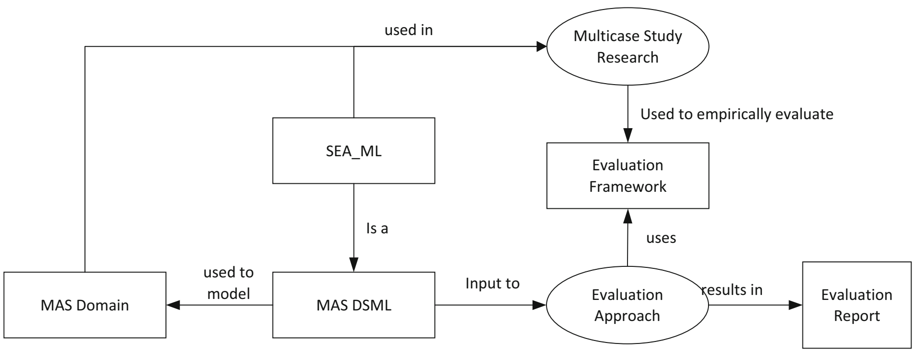

Fig. 2 Conceptual model of the DSML evaluation approach representing the context of the framework

DSML as input that will be evaluated according to the *Evaluation Framework*. Hereby, *MAS DSML* is used to model *MAS Domain* applications. As stated before we will use *SEA_ML* as an example *MAS DSML* to illustrate the evaluation approach. The result of the *Evaluation Approach* is an *Evaluation Report* that characterizes and describes the strong and weak points of the *MAS DSML*. In this paper we will adopt a *Multi-case Study* approach to assess this evaluation process. The multi-case study protocol is explained in the next section.

Table 1 presents the evaluation framework with the corresponding evaluation dimensions and criteria. The evaluation framework adopts three basic dimensions including *language*, *execution*, and *quality dimensions*. The first two dimensions and their criteria are considered as quantitative part of the assessment, while the latter one deals with the qualitative part of the evaluation. The language dimension includes the sub-dimensions of *language elements* and *model transformations*, each of which covers specific criteria that are evaluated. The language elements sub-dimension includes the criteria of *abstract syntax*, *concrete syntax*, and *static semantics*. The model transformations sub-dimension includes the criteria M2M and M2T which represent model-to-model transformations and model-to-text transformations, respectively. In essence, these criteria of the evaluation framework aim to characterize and analyze the structure of the evaluated DSML.

Characterization and analysis with the related criteria can be used for evaluating and comparing MAS DSMLs. To this end, the framework considers different aspects of MASs in the abstract and concrete syntaxes to examine capabilities of MAS DSMLs in modeling the agent system. These aspects are called viewpoints and addresses MASs from various perspectives such as Agent Internal, Plan, Protocol, Interaction, Role, and so on. Although these viewpoints are well-known in MAS domain (e.g., Hahn 2008; Hahn et al. 2009; Challenger et al. 2014), they can be extended with other viewpoints covering static and dynamic issues of MAS models (e.g., Beydoun et al. 2009). These assessments of abstract syntax and concrete syntax from various viewpoints can pave the way to compare MAS DSMLs in a higher level of abstraction and in a domain-specific way. While analyzing abstract syntax based on this framework can demonstrate the expressiveness of the DSML, analyzing the concrete syntax shows the capability of the language tools. For example, considering agent internal viewpoint of two MAS DSMLs, the one with more meta-elements and detailed relations in this viewpoint of its metamodel may have more capabilities to model the internal structure of the agents. Also, by analyzing this viewpoint, we can conclude that which MAS DSML supports BDI, Reactive, and/or some sort of hybrid agents; hence, it makes the evaluation and comparison possible from this perspective. A similar analysis can be performed for Plan viewpoint to show how much powerful the modeling ability of the MAS DSML is in elaborating the planning of the agents, e.g., the planning supports Hierarchical Task Networks (HTN), behaviors and/or primitive tasks. This scenario is also true for the other viewpoints regarding evaluating and comparing the MAS DSMLs.

The framework not only provides the criteria to evaluate the syntax of MAS DSMLs, but also enables to evaluate the semantics of them by considering their static semantics, and model transformations. For the static semantics, the number and types of MAS domain controls can be considered; e.g., whether the MAS DSML can check the agent protocol based on the MAS principles. For the M2M transformations, we can examine the number of the rules and the coverage of these rules in the target MAS domain, e.g., whether the transformations can support different agent architectures and different agent platforms. For the M2T transformations, the number of code generation rules/templates can be considered in addition to how many target agent platforms can be addressed to generate their

Table 1 Evaluation framework including its dimensions and criteria

Evaluation dimension	Evaluation sub-dimension	Evaluation criteria	Description
Language dimension	Language elements	Abstract syntax: Metamodel for the viewpoints such as Agent internal, Plan, Protocol, Interaction, Role	The number of abstract syntax model elements including views, meta-elements, attributes, connections, and so on
		Concrete syntax: Tooling for the viewpoints such as agent internal, plan, MAS Organization, Environment	The information about language notations and tooling including number of diagrams, nodes, links, and so on
		Static semantics: MAS domain controls	The number of constraints provided for the DSML
	Model transformations	M2M: Supporting BDI, reactive, and/or hybrid behavior models in the target MAS models	The number of rules that are necessary to perform the model-to-model transformations from source model to target model.
		M2T: Supporting the generation of executables for agent behavior models and MAS interactions	The number of rules that are necessary to perform the model-to-text transformations from platform-specific model to code.
Execution Dimension	Modeling input/output	Input MAS Model	The models, which are designed by language developers, are analyzed based on the number of their modeling diagrams, number of elements, relations, and so on.
		Target Agent Models: e.g., JADE, JADEX, JACK	Generated models from M2M transformations for the target languages.
		Generated MAS Artifacts	Automatically generated files, codes (LoC), functions/modules for the target languages
	Development	MAS Delta codes by developer (# of Plans, SWAs, SWSs and LoC)	The codes added by developers to the generated codes including number of files, LoC, functions/modules, Plans, SWAs, SWSs
		Overall (output) performance ratio	Generated artifacts ratio/generated code + delta artifacts The performance can be calculated for the generation of files, LoC, functions, SWA, SWS, Plan, and overall average
		Development time	Considering times for analysis, design/modeling, generation/development, error detection/testing, maintenance, and overall average

Table 1 continued

Evaluation dimension	Evaluation sub-dimension	Evaluation criteria	Description
Quality Dimension	General DSML assessment	Domain scope and suitability Domain Expertise and Expressiveness Effective underlying generation High-Level Abstraction Viewpoint oriented Understandability and Maintainability Modularity and Reusability Well-written and Readability	Qualitative analysis
	User perspective	How much SEA_ML made the development easier? What are the main advantages? What are the main disadvantages?	Qualitative analysis for different users: End users (MAS developers), Language developers (MAS DSML developers), MAS project managers (to select a MAS DSML)

architectural code, such as JACK (AOS 2001), JADE (Bellifemine et al. 2001), JADEX (Pokahr et al. 2005) and so on.

The *Execution dimension* includes two sub-dimensions *Modeling I/O* and *Development*. The sub-dimension modeling I/O includes the criteria *Input models*, *Target agent models*, and *Generated MAS artifacts*. The Development sub-dimension includes the criteria *MAS delta codes by developer*, *overall (output) performance ratio* and *development time performance*.

The *Quality dimension* includes *General DSML assessment* and *Users' perspective* sub-dimensions. General DSML assessment sub-dimension covers criteria for qualitative analysis of general properties of the DSML, while the user's perspective sub-dimension considers the advantages and disadvantages of using the DSML. The users consist of end users, MAS DSML developers, and MAS project managers.

As it can be observed from Table 1, the evaluation framework focuses on evaluating the language (including its elements, transformations, and general assessment), its artifacts (including modeling input and output), and the evaluators' results (including output performance and development).

Different aspects of the evaluation are considered to support concerns of various stakeholders of MAS DSMLs including MAS developers, MAS DSML developers, and MAS project managers. MAS developers are end users of the DSMLs who mostly deal with executional dimension and end user perspective in the evaluation framework. Considering MAS DSML developers' perspective, as language developers, they are mostly interested in language dimension of the evaluation including language elements and model transformation. Finally, concerns of MAS project managers cover almost all aspects of the evaluation framework (both of the other stakeholders' concerns) to consider the internal structure of the language, its artifact generation performance, and the end user's perspective. The evaluation process based on this framework is applied for the evaluation of SEA_ML using a multi-case study which is elaborated in the following sections.

It is worth indicating that the proposed evaluation framework and accompanying multi-case study protocol are partly constructed by benefiting from some existing work within this field. For instance, "ISO/IEC TR 15504-2" (ISO/IEC 1998) software process assessment standard and the guidelines offered in Runeson and Höst (2009) are followed in our study for conducting and reporting the case study research. In addition, the study for the assessment of benefits and risks of product lines given in Schmid and John (2002) is used while defining the assessment process, especially for conducting a case study protocol. Hence, both Schmid and John (2002) and Runeson and Höst (2009) are considered while building our detailed step-by-step multi-case study protocol including its preparation, execution, and analysis phases. The cost model of our evaluation, used during execution with data collection, is extended from the model given in Tekinerdogan et al. (2011) in which the authors assess their study to apply model-driven architecture (MDA) techniques on process sensitive embedded user assistance to provide the end user the necessary guidance. We extended their cost model in our evaluation framework in a way that it supports the cost of entire assessment process including different phases of initialization and development of the multi-case study. We adopted some of the qualitative criteria (such as "Domain scope and suitability," "Domain Expertise and Expressiveness," and so on) from Wile (2004), used some DSML success factors from Kahlaoui et al. (2008) and applied a qualitative assessment process similar to the one introduced in Kahraman and Bilgen (2013). Finally, regarding the team preparation and execution of the case studies, we inspired from Kosar et al. (2010) and Kosar et al. (2012) for constituting the teams, structuring and presenting the case studies, and building the questionnaire. The way of adoption and/or utilization of all these methods and techniques will be discussed extensively in the following Sects. 4, 5 and 6.

4 Multi-case study protocol

In order to evaluate MAS DSMLs systematically, we need to derive a protocol for conducting a case study research for MAS modeling languages. For the evaluation of the addressed DSML based on the proposed evaluation framework, we consider a three-phase process, as it is illustrated in Fig. 3, including preparation, execution and analysis phases. Hereby, in the preparation phase, the domain is analyzed considering the dimensions

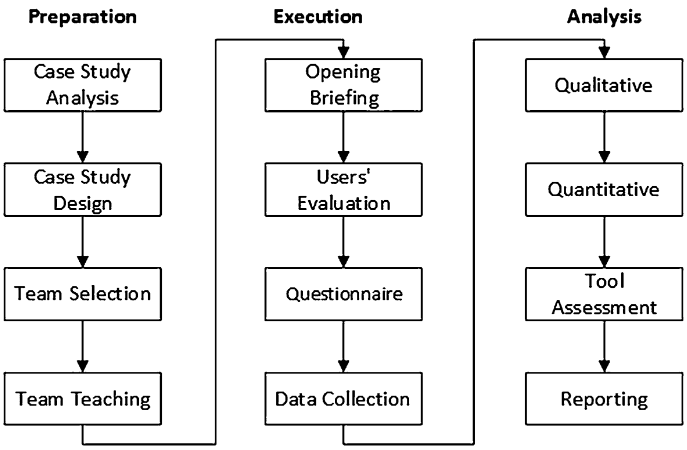

Fig. 3 The adopted process for case study preparation, execution and analysis

indicated in the framework. The domain in here is the industrial application domain of the proposed case study (stock trading, tourism, intelligent surveillance, document management, etc.) where a MAS is needed to be built. Investigation of how MAS capabilities suit well for the related domain takes place. Based on this analysis, the case studies are designed in a way that covers the dimensions specified. In order to perform the evaluation, evaluators are needed, so, the team is selected according to some criteria which will be discussed later in Sect. 4.2 of this paper. Finally, as the last step of preparation, the team is educated (tool teaching) to be able to use the language for evaluation.

In the execution phase, first an opening briefing is realized to introduce the cases (case teaching) and then the teams develop the cases. After the development, we give the evaluators a questionnaire to receive their feedback. The data are collected in this phase, as described in Table 1, including the times recorded for each phase of the development, and the data both for generated codes and delta codes.

Finally, in the analysis phase a detailed assessment report is prepared. In this report, gathered information is evaluated in detail and the final judgment of benefits and risks of performing reuse in the MAS DSML is developed. This information is extracted from the qualitative and quantitative results in addition with tool assessment and processed in a proper form, such as tables and diagrams. On the basis of these results, the report on the findings is prepared.

Based on the above process, we can state that the case studies enable to experience the usage of the MAS DSML in question and they also provide the collection of MAS DSML operational properties. After performing the case studies, some of those operational properties are compared with the similar properties of manual MAS development. Hence, this method directly supports the use of the aforementioned framework during analysis and assessment of the MAS DSML.

4.1 Selected case studies

A MAS DSML, say SEA_ML, can be utilized for different business domains when helping in the design and implementation of the agent-based systems of those domains such as e-commerce (Shih et al. 2005), document management (Pešović, et al. 2011), e-bartering (Demirkol et al. 2011) and the stock exchange system (Kardas et al. 2012). Hence, in order to evaluate the usage of the MAS DSML, design of MASs in four different domains is considered in this study[1]:

1. *Expert Finding System* Software agents in an expert finding system work on the web by using service descriptions, find candidate services, and then try to make an agreement with those services on behalf of their human users by taking into consideration QoS metrics. Services may include recommending alternative communication channels on the web, finding the nearest market to buy some special items, etc. Main goal of the case study covering the development of this expert finding system is to assess the modeling capabilities of the MAS DSML especially for the interactions between autonomous agents and semantic web services. Taking into consideration the modeling of agent internals is also crucial for this study.

[1] We keep the descriptions of the case studies short in this paper in order to keep focus on the evaluation. However, detailed descriptions of each case study, their models according to the SEA_ML specifications with including all modeling and code generation tools of SEA_ML can be found online inside a bundle at: http://mas.ube.ege.edu.tr/downloads/sea_ml_bundle.zip.

2. *Stock Exchange System* In this case study, MAS developers are requested to design an industrial software system in which Investor (Buyer and/or Seller), Broker and Stock Trade Manager agents take role in computerized stock trading. All of the user agents including investors and brokers cooperate with stock trade manager agent to access the stock market. Also, the user agents interact with each other. For instance, investor X and investor Y can cooperate with a broker in order to exchange the stock for which the broker is an expert. With this case study, MAS developers need to cope with the very complicated nature of agent behaviors in which agents pursue their own interests on trading. Conducted study helps the investigation of whether the DSML modeling features enable the easy design of such complicated interactions between agents and planning mechanism of each agent.
3. *Conference Management System* As a familiar application domain for most of the DSML evaluators who take part in this study, design of an online conference management system in which various agents act as authors, reviewers or program committee chairs during the review of papers submitted for the inclusion of a scientific conference, is performed. Dealing with the requirements and constraints of paper submission, paper evaluation, and making decision procedures, MAS developers design the requested system with the given DSML and related tools. This case study may help the assessment of the DSML features especially covering MAS organization.
4. *Hotel Reservation System* According to the preferences of their users, customer agents search online for most appropriate hotels, communicate with the hotel agents who represent hotels and make reservations if an agreement is settled with a hotel agent. In addition to being familiar to all participants, the main aim of this case study is to evaluate the modeling features considering ontology, agent-service interaction and organization viewpoints.

As can be seen from the descriptions of the case studies, each serves for a different MAS application domain with including various features of agents. In order to clearly assess the DSML capabilities on the support of agent abstractions and viewpoints, each case study also concentrates on different MAS viewpoints, e.g., agent internals, environment, service interaction, organization (as again can be seen from the above description). That makes each case study unique and own different modeling and implementation complexity level. For instance, case study 2 requires a complicated design of agent interactions and agent internals, whereas case study 4 mostly reflects the development of MAS organizations. If a grading of the design and implementation complexities for the case studies relative to each other is taken into account, case studies 1, 2, 3 and 4 can be graded as 4, 5, 2, 3, respectively, according to a 1,…,5 scale where 1 means the least complex while 5 means the most complex case study.

4.2 Team preparation

As part of preparation for the case study protocol, the evaluator teams should be prepared. To this end, two groups, including ten people, of the users/developers are selected, see Table 2. The first group is considered to use SEA_ML to design and generate/implement the code for the case studies. The other group is considered to work on the case studies manually, without using SEA_ML language (but they can use generic software tools). Each group has five members, including a PhD and four master students. Every group member has experience on software development (including MASs) in industrial scale with varying durations (4 years on the average). In fact, we aimed at organizing groups with much more

Table 2 Evaluation group profiles

Groups	Number of members	Number of PhD candidates	Number of Master students	Type of Development
A	5	1	4	Using SEA_ML
B	5	1	4	Manual (without using MAS DSMLs)

than 5 members. However, adoption of both MAS and Semantic Web technologies in real system implementations has been recently emerged and the developers especially with industrial experience on the application of these technologies comparing with any other trivial domains are very rare. So, only five people for each group were able to be gathered. Most of the team members have passed related graduate courses in their master or PhD program, including Agent-oriented Software Development, MASs, Meta Data, Semantic Web, and Web Services which are taught in Computer Engineering department and International Computer Institute of Ege University, Turkey. For those evaluators who have not passed one or more of these courses, we had a teaching session on an individual base or group form. Also, at least one member of each group works in a software company and own professional development experience with industrial MAS development for 2 years on the average.

4.3 Hypothesis

To realize the evaluation of a MAS development modeling language, we postulate a hypothesis. This hypothesis will be tested after applying the case study research based on the evaluation framework and analyzing the results. The hypothesis states that:

The performance of MAS development with employing a DSML (say SEA_ML) and its tool is more than the performance of MAS development manually or with the aid of the other non-DSMLs, such as general software development tools (e.g., UML), and other agent development methodologies [e.g., Prometheus (Padgham and Winikoff 2004)] based directly on the development by using GPLs. Within the scope of the *performance* mentioned in this hypothesis, only the following characteristics of each language will be compared in order to test the hypothesis:

1. The development time: This is one of the important cost factors of MAS development. In fact the main cost of development is programmer's time which mainly is spent for initial development.
2. The testing time: Error detection and resolving determine the cost of software testing. Software testing is another important factor of a MAS development. Clearly, the reason is that finding these errors and resolving them takes some noticeable effort. Also, it is possible that some errors lead to cascading errors which demands higher cost of testing. This is especially true when the errors are in the architectural level.
3. The maintenance effort: After system development and testing, its maintenance in the long term is one of the effort consuming issues. To deal with this issue, in this study, we have defined extensions for the case studies. The time for these extensions represents maintenance effort in this study.

4. Software quality: We consider software's structural quality which refers to how the software meets non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability, the degree to which the software was produced correctly. Structural quality is evaluated through the analysis of the software inner structure, its source code, which is in effect how its architecture adheres to sound principles of software architecture outlined.

These different aspects of development performance are considered in the proposed evaluation framework as execution dimension and are examined in the conducted case study research.

5 Execution of the case studies

Execution of the case studies starts with a session for introduction of case studies to the evaluators. This provides the adjustment of the familiarity level of all developers with the problem domain, since it can affect their development time. Then, the evaluators fulfill the development process. Time elapsed for their development activities in addition to the information of the generated and delta codes are recorded for the later analysis. This step, called data collection, is a key part of an empirical study. However, as is in any empirical study, there are some threats to the validity of the data collection which is discussed later in this section.

It is worth stating that team members worked individually for the development of the case studies and the times were recorded for each of them and for each case study separately. The average times were calculated for each phase of development considering all members and all case studies to reduce the risk of validity.

The workflow of both groups' development processes is depicted in Fig. 4. According to this workflow, each developer in group A and B should follow several steps to realize the development. To provide similar conditions for all of the developers, all of the steps are mandatory for both groups. These steps are more elaborated and analyzed in the next section where execution of the multi-case study is discussed. Moreover, meeting all requirements specified for each case study is requested from every developer in order to provide fair load of tasks.

While the group A used the MAS DSML (SEA_ML) and the APIs of target agent platforms (JADEX and JACK), the group B used the generic software development tools

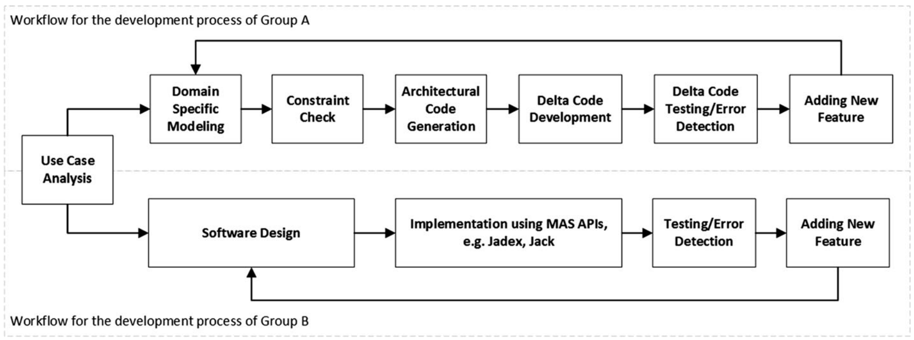
Workflow for the development process of Group A
Domain Specific Modeling
Constraint Check
Architectural Code Generation
Delta Code Development
Delta Code Testing/Error Detection
Adding New Feature
Use Case Analysis
Software Design
Implementation using MAS APIs, e.g. Jadex, Jack
Testing/Error Detection
Adding New Feature
Workflow for the development process of Group B

Fig. 4 The workflow representing steps of the development for Group A and Group B

and approaches such as UML and general methodologies for agent system's analysis and design, e.g., Prometheus (Padgham and Winikoff 2004), Gaia (Zambonelli et al. 2003), Tropos (Bresciani et al. 2004). In this regard, in group A, two evaluators selected JADEX platform and three evaluators preferred JACK. On the other hand, in group B, two developers used UML for their analysis and design, and JADE framework as the development environment, two developers used Prometheus methodology and JACK platform, and finally one developer used simply pen and paper for her analysis and design in addition to JADEX platform for implementation. Although languages (SEA_ML, JACK, JADE, JADEX) were different, both groups used the same integrated development environment (IDE), well-known Eclipse (Eclipse 2004) in order to provide fair assessment as much as possible. SEA_ML's modeling, transformation and code generation features are directly based on Eclipse. JACK, JADE and JADEX frameworks are also usable with their complete APIs over Eclipse. Hence, all evaluators used the same IDE and had similar environments for implementation with build options and appropriate editing tools. Within this same IDE, the only difference originated from the adoption of the languages as expected. For instance using the graphical concrete syntax of SEA_ML is naturally different from the text-based JADE. This difference enabled us to analyze the impact of using various languages (inside the same IDE) which in fact, was one of the goals of this study.

5.1 Data collection

In this section the steps of data collection are discussed.

5.1.1 Procedure of data collection

The data collection is done during preparation and execution phases of the case study research. During the preparation phase, some data is collected for team teaching. In this part of the preparation, times for learning the DSML, reading the documents, and running a demo are recorded to be used in the evaluation. These time data are solely collected for the group of the evaluators (group A) who use the DSML in their development. The team teaching is fulfilled for this group as a whole at once. Also, this time is needed to be spent just once for each new group in the future.

After the initial team teaching session for developers in the preparation phase to adjust the level of knowledge for the language, the execution phase starts with an introduction to the case studies called opening briefing. The aim of this part is to normalize the level of all developers' familiarity with the problem domains, since this can affect development ability for that domain which is a validity threat.

For execution of the case studies, the two groups developed the MASs for the related case studies, one with SEA_ML and one without it. During the development, times for different steps of development are recorded. Considering this timing data which is used in the performance evaluation of SEA_ML, we collected the time periods, used by the developers to perform analysis, modeling, implementation, testing, and maintenance for the case studies as they are indicated in the evaluation framework (Table 1).

After development, some other data are collected both for each case study and each developer related to the provided code. This data include modeling data and generated artifact data, see Table 1. For the modeling data, we collected the detailed information about different viewpoints of models, e.g., number of elements and relations. These data are used in SEA_ML's quantitative analysis (language dimension in the evaluation framework). Considering the artifacts, we collected the data for the generated and manually developed artifacts for each case study (executional dimension in the evaluation framework). These data can be used to compare the two DSMLs for developing the system.

Finally, to consider the evaluators' experience in the evaluation, we provided a questionnaire for them. Based on their replies, we realized qualitative analysis of the DSML (quality dimension of the evaluation).

5.1.2 Execution with data collection

As mentioned in the previous section, the data are collected in preparation and execution phases. According to the evaluation framework, to collect data during the preparation, the total initial cost is calculated based on the time spent for teaching, reading the documents, and running the demo. So, the total initialization time is calculated as follows:

$$C_{\text{TotalInit}} = C_{\text{Teaching}} + C_{\text{ReadingDocs}} + C_{\text{Demo}}$$

The total cost needed for initial preparation consists of the time for learning SEA_ML, reading the documents and running the demo by the developers. The elements constituting the total initial cost are elaborated as follows:

5.1.2.1 Learning SEA_ML ($C_{Teaching}$) In this part of the evaluation the evaluators attended in a presentation session on SEA_ML. This time is only spent with the group working with SEA_ML and the group which works with the classical approach does not spend this time.

5.1.2.2 Reading the documents ($C_{ReadingDocs}$) The evaluators who worked with the DSML are also given a manual (SEA_ML documentation) in 18 pages long. They read and understood the document. Later, they tried different steps of installation and configuration of SEA_ML as well as adding various plugins and loading required projects.

5.1.2.3 Running demo and analyzing it (C_{Demo}) A previously prepared example was provided in SEA_ML for the evaluators to learn the language. The example is an electronic bartering system, E-barter. Interested readers may refer to Demirkol et al. (2011) for the agent-based e-barter system development. Evaluators analyzed this example and run it. The time for analyzing and running this real example is recorded for evaluators. Inspiring from this demo example, they also wrote and run a hello world example based on the documentation.

On the other hand, as discussed in Sect. 4, to realize the evaluation based on the end users' development, four case studies were provided to software development groups (with or without using the DSML) to develop MASs. The times for different phases of developing the software were recorded for analysis later. Based on the following adopted general cost model ($C_{\text{TotalDevelop}}$) for developing MASs (introduced in the evaluation framework), we have provided the evaluation of SEA_ML:

$$C_{\text{TotalDevelop}} = C_{\text{Analysis}} + C_{\text{Modeling/Design}} + C_{\text{Implementation}} + C_{\text{ErrorDetection}} + C_{\text{Maintainance}}$$

The costs constituents of this cost model are explained below:

5.1.2.4 Problem analysis ($C_{Analysis}$) This is a common step, for both groups and elapsed time depends on the complexity of the problem domain of the related case studies. This time includes the time spent in reading and understanding the description of the case study (1–2 pages) and the time for analyzing the problem.

5.1.2.5 Modeling/design ($C_{Modeling/Design}$) Both of the groups spend this time, but the group working with the DSML has two advantages. First, using the language they have a big picture of the system specific to the domain and using the domain concepts and relations which accelerates the design. Second, using the language, the developer can reuse partial/ entire model in further system developments. Therefore, this time is more important for group B whose users do not use SEA_ML. Group B members, in addition to agent platform APIs, can also use generic software development tools which are not specific to the MAS domain, such as UML for their analysis and design. Also, they can use general agent development methodologies such as Prometheus (Padgham and Winikoff 2004), Gaia (Zambonelli et al. 2003), and Tropos (Bresciani et al. 2004). However, these developers can also use just pen and paper to make their design and realize their implementation based on their design if they wish.

5.1.2.6 Implementation ($C_{Implementation}$) One of the main steps of software development is the implementation phase, which can be improved by DSMLs. DSMLs generate architectural templates and interconnect the components. In this way, DSMLs can produce a noticeable part of the code. So, the methodology, equipped with a DSML, takeovers the classical approach in this point. Considering this fact, the time spent in this phase is expected to be less for the group using the DSML. On the other hand, the users using the language (group A) need very little time for generating the code by SEA_ML plus completing the code manually, while the developers without using the DSML spent longer time for cumbersome initial architectural code development with their interconnections, in addition to the complementary code to finalize the implementation of the case studies.

5.1.2.7 Error detection/testing (C_{Error}) The code generated by SEA_ML is a template code, and the relations are established by the language considering the user model. The models are controlled by the language in semantic checking phase of the language by OCL (2012) rules which prevents having errors. So, the models have no errors at this level. Also, the programing experience shows that in complex systems, developers make mistakes in the architectural level, which leads to many changes later on, costing lots of time, effort and money. So, in this manner, the methodology equipped with DSML also takeovers the classical software agent development approach in which development is done without using domain-specific modeling tools and languages. However, it still needs some error-proofing for those parts of the code which were added by the developers manually to complete the code (which is smaller part of the code and it is in common with the group B).

5.1.2.8 Maintenance/extension ($C_{Maintainance}$) To consider the process of software development after the implementation phase, we examined the case studies with modeling by adding a new feature (e.g., A SWS or SWA to the system) after it is completed. We measured the time spent in accomplishing these additional tasks for both groups (w/o DSML).

6 Results and analysis

As an empirical study, a DSML for MAS is evaluated and there is no single set of outcomes. The results can be observed from different perspectives. In this section, these results are demonstrated and analyzed. Please note that since the fractions are not logical in the code or time, the digits are rounded which also makes the comparison easier. The results related to language and executional dimensions (from evaluation framework) are discussed in Sect. 6.1, and the quality dimension of the evaluation framework is elaborated in Sect. 6.2.

6.1 Quantitative analysis

In this section the quantitative analysis is realized including the language assessment of the framework and development evaluation for executional dimension. For the first one, we consider the language elements, language's transformation and artifacts. In the latter one, development evaluation, the results gained from the automatic generation for the case studies are used for evaluating the performance of the development. In addition, the results of the case study research are also used for time performance of the development. These results are analyzed and illustrated in proper forms.

6.1.1 Language analysis and characterization based on quantitative properties

The SEA_ML language is assessed in two ways to cover language dimension of the evaluation framework. First, we analyzed the language, as a DSML, and its elements. This can show the sophistication and the structural complexity of the language and can be used in the comparison of the languages. Second, the input and output of the language are analyzed which paves the way to assess the generated artifacts.

Table 3 shows the characterization of the SEA_ML based on the evaluation framework dimensions. In this table, the main types of elements with their quantities are given for the three major definition elements of SEA_ML, regarding abstract syntax, concrete syntax, and transformation. As it can be seen in Table 3, the viewpoints in the abstract syntax are represented with the same number of diagrams in concrete syntax. Similarly, Aggregation/Containment relations and Attributes/Properties are converted to the same number of Compartments and Attributes, respectively. On the other hand, Meta-Elements and Association/Reference relations are expressed with Nodes and Links, respectively; however, their number is reduced due to some super-class elements and inheritance relations, used in the metamodel, are not directly represented in the instance models. On the contrary, well formed-ness rules are represented with constraints in concrete syntax which are not possible to be controlled in the abstract syntax level.

The data in Table 3 can be used to show the specification of SEA_ML and its tools which by itself can be utilized in its comparison with other MAS DSMLs regarding the features and/or complexities of the MAS DSMLs. As a general example, the metamodel of ACSM (Odell et al. 2005) contains 8 meta-elements (as its abstract syntax), while SEA_ML abstract syntax includes 67 meta-elements which shows the detailed modeling capability of SEA_ML. As a more specific example, SEA_ML has 57 nodes in its concrete syntax (tooling part) from which ten nodes are used for modeling the agent internal viewpoint, while a MAS DSML called DSML4MAS (Hahn 2008) has only five elements for modeling the agent internal viewpoint. As a result DSML4MAS probably cannot fully support modeling agent's belief and goals as main part of BDI agent architecture, in addition to agent's details such as agent state and agent type. So, the capability of modeling the agent's internal structure is higher in SEA_ML than DSML4MAS which paves the way to generate both more detailed target agent platform models and more detailed artifacts such as codes for the MAS domain.

Model transformations, both model-to-model and model-to-code, are the heart of MDD, and they are among the main elements of any generative DSML tool. So, in this evaluation

Table 3 The specification of SEA_ML

Language dimension	Abstract syntax (metamodel)							
Specification feature	Viewpoint (sub metamodel)	Meta-Element (Class)	Attribute (Property)	Association (Reference)	Aggregation (Containment)	Inheritance (Super Type)	Self-Reference	
Number	8	67	83	38	21	18	3	
Language dimension	Concrete syntax (graphical language)					Transformation and generation		
Specification feature	Diagram	Node	Attribute	Link	Compartment	Constraints	M2M Rules	M2T Templates
Number	8	57	83	19	21	23	50	40

we pay specific attention to them. Considering the transformations in SEA_ML language, three types of ATL rules are used for different viewpoints. One for the multi-agent part of the system (JADEX agents) which adds required SWAs into the previously generated Agent Definition Files (ADFs) of the JADEX agents, another for the SWS part of the system (OWL-S metamodel) which helps to generate OWL-S documents, and finally the last type of rules provides the transformations for generating ontologies. Altogether, 50 ATL M2M transformation rules and 40 MOFScript M2T transformation templates are provided in SEA_ML (Table 3). By applying these rules and templates, ADF files and plan files are generated when considering the modeled agents. Also, a set of OWL-S files (considering various modeled SWSs) and WSDL files (for each SWS) are generated.

To evaluate SEA_ML language, we also analyzed the input models and generated artifacts (see Table 4). Input model consists of the data which are provided for building the models. The generated artifacts include generated intermediate models and generated final artifacts. The intermediate models are platform-specific models and its components which are generated automatically from the platform-independent input models. The generated final artifacts are the architectural codes and components achieved from the intermediate models. Table 4 also includes the data related to the codes and components which are added to the generated artifacts, a.k.a delta codes, to make the software complete. These data are representing the capability of the DSML and are used in the next section to assess the SEA_ML's performance. Briefly, they can guide developers in the assessment of how much the language is powerful regarding platform-independent to platform-specific transformation. In other words, the strength of the DSML in realizing MAS in different agent platforms can be evaluated.

The information provided in Table 4 is calculated from the average for all of the case studies and all of the developers. For example, the LoC for Generated Final Code in Table 4 indicates the average number of LoC which are generated for the case studies of all of the users in Group A (who used the SEA_ML).

In conclusion, the data provided in Tables 3 and 4 can be used to compare and evaluate the performance of a MAS DSML. To this end, the detailed data about abstract syntax of the DSML in Table 3 can be used to compare the expressiveness of modeling language (e.g., extensive support for agent viewpoints, transformability for real MAS execution platforms) for MAS domain. For example, if number of elements and relations related to Plan viewpoint of the abstract syntax of a DSML is more detailed than of another DSML, this will show the ability of Plan modeling for the first DSML. In a similar way, the data for concrete syntax and model transformations can be used to compare MAS DSML regarding the ability of tooling for the MAS domain, and the ability of covering the target domain model in addition to target framework final code.

On the other hand, the data presented in Table 4 can pave the way to calculate the performance of the MAS DSML. In this regard, the details related to input models and intermediate models can give the notion of the size of case studies in the evaluation. These data are important to validate the performance evaluation. For example, in the case of comparing the performance of two MAS DSMLs, their source models show that whether the performance is calculated in the similar size or not (otherwise the performance comparison can be invalid). In addition, intermediate models help to assess the coverage of MAS target model for the MAS DSML. Finally, the data related to generated artifacts and delta codes in Table 4 are used to calculate the development performance of a MAS DSML as part of the evaluation based on the formula given in the next section.

Table 4 The specification of I/O and Delta Code for SEA_ML

Element	Input model				Generated intermediate models						Generated final artifacts			Developer's delta code					
Feature	Model diagram	Element	Relation	Attribute	SWA (ADFs)	Plans	SWS	Interface	Process	WSDL	File	LoC	Function/module	File	LoC	Function/module	Plans	SWA	SWS
Number	8	138	153	170	10	12	2	3	3	2	32	925	47	2	216	15	2	2	0

6.1.2 Performance analysis through the multi-case study

In this section the development capability of SEA_ML is assessed. This evaluation is realized considering the executional dimension proposed in the framework and provides two types of performance evaluation: (1) Development performance (generation performance) regarding the ratio of overall generated codes and artifacts to those which are manually developed. (2) Performance of development time regarding the time which is spent by the developers using SEA_ML and those who do not use it.

For calculation of the development performance of SEA_ML, we compared the automatically generated code with the code which was added by developers of group A to complete the code. Then the performance was calculated based on the percentage for different artifacts which are generated and manually developed, such as files, LoC, SWAs, and so on. The percentage shows the rate of generated or manually added artifacts comparing to the total number of the artifacts. To this end, we use the information provided in Table 4, generated artifacts and delta code, and provide Fig. 5.

The performance calculation was realized using the following formulas for each artifact:

G_X	Generated number of artifact X
M_X	Manually developed number of artifact X
T_X	Total amount for artifact $X = G_X + M_X$
Pg_X	Performance for generated part of artifact $X = (G_X \times 100)/T_X$
Pm_X	Performance for manually developed part of artifact $X = 100 - \mathrm{Pg}_X$

In Fig. 5, rate of generation for the most important components of generated artifacts is considered. The highest rate of generation is with SWS for which all of the desired ones are modeled and generated. Although no new SWS was added after modeling the system by developers, they added new Interface, Process, and/or Grounding for those SWS.

On the other hand, generated functions/modules have the least rate of generation in Fig. 5. This is because the personal interests' of the developers in using the number of functions in the program affects this factor. In other words, some of the developers like to divide the tasks in tinier modules and write more functions, when they complete the code.

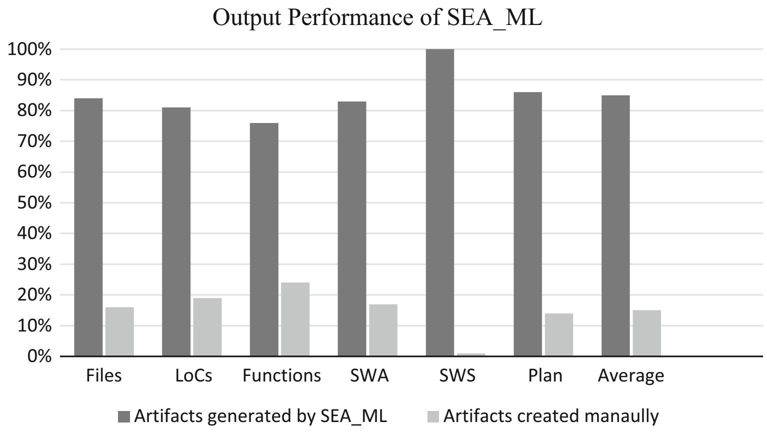

Fig. 5 Output performance of SEA_ML

Generally, considering the results provided in Fig. 5, we can see that for average developers and case studies, 85 % of the artifacts are generated and 81 % of the whole code is generated.

Finally, the results show that the use of a DSML for MAS working on the Semantic Web can reduce the complexity of developing these MASs considering both the complexity in the MASs (various plans, roles, goals, beliefs, and capabilities) and the complexity of Agent's interaction with each other and with SWSs. This is done by generating many artifacts required for developing the MASs and providing their interconnections.

On the other hand, one of the key concerns of the developers using the language is to provide the software in a cost-effective manner. To evaluate the cost (i.e. time) of using the DSML, both the time of preparation for the groups and the time for development are considered.

The results of the preparation are provided based on the times recorded in the execution phase of the case study, and calculations are carried out using the formulas presented in the previous section. These data items are collected from group A developers and the average elapsed time is calculated, as shown in Table 5.

In fact, Table 5 shows the initial overhead cost of using the DSML. To use the language, one should spend this time and effort to prepare the developers to be able to use the language and take the later benefits. This additional average time of 7 h 45 min is not required for group B. They only need to be familiar with MAS and Semantic Web programing, which constitute the domain knowledge of this study and having this domain knowledge is a must for each developer. Although we had separate sessions for each of the developers who has weakness in any of these domains or their programing, we did not consider this time in the evaluation since that is not part of the evaluation of the language.

Different case studies with two different groups were considered for the development time evaluation. The results for the criteria mentioned in the evaluation framework are collected as described in the previous section, and the averages of these results are calculated for each team (five people). These results are shown in Table 6 for all case studies.

The average result of each group, including five members, is also shown in Table 6. To gain a general result for all case studies, the average for them is calculated as the last row of Table 6. In other words, these data show the average of each item, e.g., analysis, modeling, and so on, for five developers and for four case studies; separately calculated for each group. In this way, the results can be compared easier. To focus on the difference, we also demonstrated the average row in a bar chart diagram depicted in Fig. 6.

To analyze the result, we compare the results for groups A and B based on Fig. 6. By considering the results of group A and B for each item, we can find out the following conclusions:

- Analysis of a problem is part of the development of any software system. For analyzing the case studies, both groups spent more or less the same amount of time. In case study 1 and 3, group A spent slightly less time and in the other case studies, group B spent

Table 5 Initial preparation costs

Group	C_{Teaching}	$C_{\text{ReadingDocs}}$	C_{Demo}	$C_{\text{TotalInit}}$
A	2 h:30 min	3 h:30 min	1 h:45 min	7 h:45 min

Table 6 The result of the average total cost and its elements in two groups and four different case studies

#	Case study	Team	C_{Analysis}	C_{Modeling}	$C_{\text{Implementation}}$	C_{Error}	$C_{\text{Maintainance}}$	C_{Total}
1	Expert finding	A	1 h:36 min	0 h:43 min	0 h:34 min	0 h:15 min	0 h:31 min	3 h:39 min
		B	1 h:40 min	1 h:11 min	2 h:53 min	0 h:34 min	0 h:54 min	7 h:12 min
2	Stock exchange	A	1 h:47 min	0 h:53 min	0 h:44 min	0 h:21 min	0 h:24 min	4 h:09 min
		B	1 h:39 min	1 h:19 min	2 h:50 min	0 h:42 min	0 h:51 min	7 h:21 min
3	Conference management	A	1 h:07 min	0 h:40 min	0 h:37 min	0 h:13 min	0 h:21 min	2 h:58 min
		B	1 h:13 min	1 h:03 min	1 h:52 min	0 h:31 min	0 h:47 min	5 h:26 min
4	Hotel reservation	A	1 h:18 min	0 h:47 min	0 h:32 min	0 h:17 min	0 h:28 min	3 h:22 min
		B	1 h:06 min	1 h:12 min	1 h:40 min	0 h:29 min	0 h:49 min	5 h:16 min
5	Average	A	1 h:27 min	0 h:46 min	0 h:37 min	0 h:17 min	0 h:26 min	3 h:33 min
		B	1 h:24 min	1 h:11 min	2 h:19 min	0 h:34 min	0 h:50 min	6 h:18 min

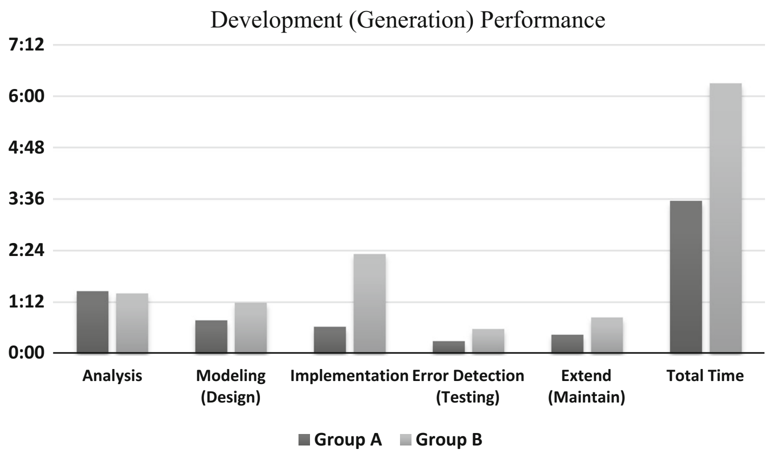

Fig. 6 The diagram comparing the general average for development time of Group A and B

less time than group A. The reason is that this step is independent of the development language and is only dependent on the problem complexity.

- The modeling time is close for both groups, but group A is a bit faster. The reason of this difference is that we asked both groups to do their design before implementation; and the group A could use the DSML and work with the domain concepts in design time, while group B fulfilled this step without SEA_ML. However, group B could use any general-purpose modeling language such as UML in this phase.
- Implementation is the takeoff point in the comparison of the groups. The time for implementing the case studies for group B is by far more than the time for group A. Clearly, the reason is the DSML. A large amount of the files, code and their interconnections are generated automatically by the language, while group B realizes all of these coding and interconnecting manually. Although we saw slight improvement in the later implementation of group B members considering the time of coding (due to mastering on the code), this amount of improvement can be overlooked regarding its small size in comparison with the whole time of implementation. Another point is that, even though the architectural code is generated by the language, group A still needs to complete the code to have an executable program, e.g., filling the plans to be run by an agent.
- In error detection/testing phase, both groups try to find the syntax and semantic errors of the programs. Group B finds the errors in the whole code, while group A finds only the errors in the added complementary code which is by far less than the generated code. The reason is the architectural generated codes, which are majority of final code, consist of error-free ready template codes which do not need to be re-validated. The models from which the codes are generated are validated not only by constraint check (Challenger, et al. 2014) but also can be formally checked as discussed in our previous work (Getir, et al. 2014). The time for this validation and model checking is considered as part of the total modeling time. So the developers of group A only need to check their delta codes for error detection/testing after development, as it is illustrated in the development process workflow (Fig. 4). As a result, the amount of errors and the time needed to resolve them for group A is less than of group B.

- To evaluate the effect of the DSML after the development, an extension is asked from each developer (of both groups) for each case study. Group A uses the DSML for this extension. In this way, they add or modify the instance models and re-generate the code. However, group B makes this modification manually. Therefore, the time for group A is less than of group B. Since the amount of the code needed in this step is rather small, the difference in the time spent in developing this extension is relatively small. Another reason for the small size of this difference is the fact that group A deals with understanding the newly generated code to find where to add the code related to the extension, since the language lacks round-trip development needed for iterative development.
- In conclusion, the total time of development for both groups shows that group A needs half of the time group B needs for developing the case studies in average.

Hence, the quantitative results are provided for the evaluation of SEA_ML based on the proposed evaluation framework as part of the analysis phase (Fig. 3). To this end, the result of the language and its tool assessments are reported in Tables 3 and 4. Also, the development performance of SEA_ML is presented in Fig. 5 by analysis of the MAS DSML artifacts.

6.2 Qualitative evaluation

SEA_ML is evaluated according to the criteria previously introduced for the quality dimension given in Table 1 of Sect. 3. This evaluation is performed by taking into account two criteria. First, the general assessment is fulfilled by discussing the general pros and cons of the proposed DSML. Second, feedbacks are gained from the users, in this case the developers, who have used the SEA_ML and its language in their developments (group A). To this end, a questionnaire is filled by each developer of group A.

6.2.1 General assessment

In this part, we evaluate our DSML, considering some of the selected success factors presented in Kahlaoui et al. (2008), Tekinerdogan et al. (2011), and Kahraman and Bilgen (2013) to provide a general assessment.

6.2.1.1 Domain scope and suitability Domain scope of a DSML is very important, because it determines the utility and usefulness of the DSML. If the domain scope of the evaluated DSML is too broad, DSMLs will be less specific and less expressive; if it is too narrow, the return on investment might be low (Kahlaoui et al. 2008). Agent developers who especially intend to use SWS are targeted in SEA_ML. So, SEA_ML domain scope is narrow enough to fit the needs of the end users. Also, the addressed domain should have enough structural complexity to be targeted with increasing its abstract level (Mernik et al. 2005). The domain is suitable for applying DSML approach considering its internal and external complexities while developing MASs. Each agent has its own roles and capabilities, including plans, goals and beliefs. These components work with each other and have interconnections which lead to agent's internal complexity. Also, the agents have interactions with other agents and with SWSs which lead to the agent's external complexity. A DSML can model these interactions and generate the code from these models and reduce the complexity by increasing the abstraction level.

6.2.1.2 Domain expertise and expressiveness Starting with conceptual independent model, we extracted detailed elements and relations for the system. These elements belong to different perspectives of the system. Richness of the domain concepts, vocabulary and terminology increases expressiveness and functionality of our DSML.

6.2.1.3 Effective underlying generation Having an automated generative mechanism for a DSML makes it to be more than an exclusively documentation tool. SEA_ML can generate complex code schemas for OWL, OWL-S, and JADEX.

6.2.1.4 High-level of abstraction We demonstrated high abstraction level of the proposed metamodel by transforming it into different agent technologies [JADEX (Pokahr et al. 2005) and JACK (AOS 2001)], as well as SWS technologies such as OWL-S.

6.2.1.5 Viewpoint oriented Using different viewpoints, the DSML is specialized and different aspects of the domain are separated which makes its usage and comprehension easier (Tekinerdogan and Demirli 2013).

6.2.1.6 Understandability and maintainability As it has been described in the literature (Hannemann and Kiczales 2002; Garcia et al. 2005; Monteiro and Fernandes 2005), the invasive nature of pattern implementations with the base code may reduce the modularity of the system, thus affecting its understandability and maintainability. In the contrary, a DSML, as a MDD approach, manages the templates using a model for the software system which leads to increase understandability and maintainability. Also, the commonly held belief that DSL programs are easier to understand and maintain than GPL programs had previously been empirically validated in Kosar et al. (2012). In case of SEA_ML, the tool itself is quite easy to understand since the software developers have in effect to define a kind of graphical diagram for the different parts of a MAS. Since the notion of diagram was well-known in the domain, it was not difficult to define the MAS.

6.2.1.7 Modularity and reusability As the MASs consist of several concerns which are elaborated in different viewpoints, the targeted software systems are modularly integrated in the models. In case of required changes to one of the concerns, this can be easily located and adapted to the new requirements by updating the aspects. All the model information is stored in XML files which can be easily accessed and reused to define new MASs. Through defining viewpoints and automating weaving the concerns in the final application, the complexity of the overall development can also be reduced and maintenance is substantially improved with respect to the manual approaches. In addition, due to the modularity of models, it is possible to reuse the DSML artifacts and/or tools in multiple projects in order to compensate for the high initial cost of developing tools and training human resources. Similar conclusions were made in Kirstan and Zimmermann (2010) based on the interviews with companies in the car industry where they reported that applying the paradigm alone does not lead to cost and time savings without reuse.

6.2.1.8 Well-written and readable Using symbols that are too close to domain leads to remove personal definition and interpretation of terms and jargon. This is an appealing feature of DSMLs which avoids ambiguity in the models for the domain problem. The result is the increase of models readability which is very important for programing languages (Khedker 1997). In other words, the models are well-written and more readable which makes them appropriate for audience. We have not received any complaints for the readability of the models from our evaluators in their questionnaires. We believe that the main reason is the appropriateness and completeness of concepts provided in SEA_ML for MAS domain (Suitability).

Finally, it is worth indicating that DSML development has a cost overhead one time at the beginning of the methodology compared to the classical methods. Also, using DSMLs needs a learning time overhead (only once for each team) compared to the classical approaches. However, these overheads are covered in the later phases of software development, including the implementation, error detection, and maintenance of the system; where they happen repeatedly for different system developments. Also, in the long term, the time and money saved via reuse is the matter of issue in the later developments.

6.2.2 Questionnaire

To establish the qualitative evaluation based on the end user's perspective, a questionnaire is provided for the group A, who used the DSML and wrote some complementary code (delta code) for the case studies. In this way, we examined SEA_ML and its pros and cons from the user's point of view. To do this, the questionnaire is provided including following three main questions:

1. How much SEA_ML made the development easier? (Point out of 5; 0 is nothing and 5 is at most level)
2. What is (are) the main advantage(s) of the language and applying it?
3. What is (are) the main disadvantage(s) of the language and using it and suggestions for them?

According to the results, in average, they marked 4 out 5 to question 1. In fact, grading the use of SEA_ML for easiness was naturally subjective and directly based on each evaluator's own comparison between using a MAS DSML and applying a classical development methodology without using a DSML. However, we think that such self-assessment and grading SEA_ML in this way are convenient and acceptable since evaluators in group A possess substantial experience on designing agent systems with using classical software development approaches (different from MDD) and implementing MASs by utilizing only GPLs. Hence, they can perceive the enhancements brought by DSML usage (if they think that those improvements really exist) and are capable of resolving whether MAS DSML such as SEA_ML facilitates the development or not.

On the other hand, we categorized the replies for the other two questions as follows:

6.2.2.1 Advantage(s)

1. The language is easy to use, basically because it has a graphical drag and drop palette.
2. It saves the time in the development phase by generating the code.
3. The language is also helpful for brainstorming the ideas in the design time by providing main concepts and their relations from which a partial product will be produced.

6.2.2.2 Suggestion(s) There is a need for transforming the changes in the generated code back to the models (round-trip engineering) to complete the iteration of software engineering. In this way, when the user extends the models, he/she can keep the manually added code embedded in the newly generated code.

Above-mentioned items are the issues which have been put forward by the majority of the evaluators. In other words, those are the main pros and cons of proposed methodology using SEA_ML. However, there are some other points which are suggested by a group of the team members. In this part, we also report those ideas especially exposing the benefits of using SEA_ML during MAS development.

Evaluators believe that the language covers both BDI and reactive architectures for agents. This offers more flexibility for the developers while designing a MAS. Also, the evaluators believe that SEA_ML's metamodel provides detailed enough elements and relations considering Agent-SWS interaction viewpoint. This is one of the challenging parts of designing MASs working in Semantic Web environment which is abstracted by SEA_ML.

On the other hand, the evaluators had some suggestions for shortcomings of the proposed methodology using SEA_ML. One of them was to have ready-to-use agent protocols inside the SEA_ML. Although it is easy to provide any agent protocol using the language, some of the evaluators suggested that it is useful to have some of the well-known protocols, such as Contract-net protocol (Smith 1980), built-in the SEA_ML. They believe that this kind of protocols is used frequently and having them in-hand can help the developers and save their time, or even prevent some possible errors. For example, ready-to-use template models of some well-known agent communication protocols can be included in SEA_ML's distribution. So, this issue is considered as one of our extensions for SEA_ML.

Some of the evaluators think that although SEA_ML successfully help designing the system statically, it would be much better to extend the capabilities of dynamic modeling specifically for the agent behaviors. In fact, SEA_ML already supports model checking and validity of both agent behavior models and dynamic structure of agent-service interactions as discussed in our another work (Getir et al. 2014). Required semantic checking is realized based on the designed models and by using predicate logic and model analyzer of Alloy (Jackson 2002) which is a well-known declarative language based on first-order logic to define complex structures and behaviors of systems. In this way, the controls are automatically completed for the instance models before they are used to produce the artifacts. However, in order to enable model checking of SEA_ML instance models, currently a developer needs manually to convert its MAS model conforming to Ecore into the appropriate serialization required by Alloy environment. Automatization of that conversion would extend SEA_ML's capability and hence meet the expectation of those agent designers. But, an automatic and complete conversion between Ecore and Alloy is missing at the time of our study conducted.

6.2.3 Hypothesis test

To test the hypothesis, we examine its items which are discussed in the hypothesis proposal. In this way we manifest the performance of SEA_ML. So, the validity of the proposed hypothesis is checked. To this end, we analyze the items one by one as follows:

1. The development time: In fact, this time includes analysis, design and implementation of the system. Considering Fig. 6 and the discussion presented in the analysis section, the total time of development for Group A is less than of Group B. So, the performance of MAS development using the modeling language (Group A) is more than manual development (Group B).

2. The testing time: We evaluate this time by considering the time which is spent in finding errors and resolving them. As you can see from Fig. 6 and its analysis for testing, the time used for testing for Group A is less than of Group B, which reflects a higher performance for Group A, who are using the language. Since the cases are rather small compared to the real industrial cases, the number of errors is small. Therefore, for the bigger cases with more components and probably more errors, this difference can be bigger.
3. The maintenance effort: Maintenance is considered with amount of effort used for later extensions of the application. To emulate this, in our case study research, we have an extension for each case study which is applied by each evaluator. The time for applying these extensions represents the maintenance effort in our evaluation. Based on Fig. 6, the effort used with Group A is much less than of Group B which demonstrates the higher performance of the DSML compared to other non-specific tools. Also, as it is discussed in the previous item, the difference between the performances of the two groups will be bigger in the real industrial cases with a bigger scope size.
4. Software quality: Based on the questionnaire, the automatically generated code includes a better architecture. Also, different patterns are used in the generated code which increases the quality of the software.

As we can see from the abovementioned scope analysis for the hypothesis, each scope has improvement by applying the proposed development approach using SEA_ML. So, the performance of MDD in our case using the proposed methodology is higher than the one for classical approach. Therefore, the assumed hypothesis is true.

7 Threats to validity

As it is the case in any evaluation study, there are also some risks and threats to the validity of this study. The threats can be originated from different validity types including construction validity, internal validity, and external validity. Construct validity refers to what extent the operational measures that are studied really represent what the researcher have in mind and what is investigated according to the research questions. Internal validity relates to a causal relationship between treatment and the outcome. External validity concerns the ability to generalize the results of the study.

Since the domain of agents interacting with SWSs is not an established professional occupation field, we could have limited number of researchers in the evaluation. Such professional evaluators are familiar with the concepts and their relations which makes the development and subsequently the evaluation more real. Because of this shortcoming, we have a small size society for the evaluation. So, we have not considered statistics, such as *t* test, for the questionnaire considering the size of the society. This may bring the risk of validity for the results in the related part which is a conclusion validity. However, we have selected the participants randomly from the available society. Finally, we have summarized the findings for development teams and interpreted them in the analysis section.

Also, the small sample size is a threat to the generalization of the development and questionnaire results to a larger population which is an external validity. However, we do not generalize the findings and only consider the findings as outcomes of case studies.

On the other hand, the domain is under research and there are not many industrial uses for the large cases by companies. So, our evaluators are not too experienced in the case studies' domain, although they are familiar with MDD. Using such industrial real case studies could improve the results, as they propose real problems in the industry which is solved with this technology and the evaluators could be familiar with the domain's real cases. To mitigate this risk of construction validity, we have provided several case studies from various viewpoints aiming a high coverage of domain application and comprehensiveness for the evaluators.

Considering the threats of validity in the execution phase of the research, case studies can be realized with a combination of single group/two groups and one problem domain/two different domains. Using a single group has the advantage of having the same level of knowledge for solving two problems; however, working on the first problem helps the user in working on the second problem and this affects the result of evaluation (another internal validity). On the other hand, using two groups eliminates this risk, but controlling the level of knowledge and experience of two groups to have them in the same range is difficult and can strongly affect the result of the evaluation. In this study, we have two groups and the evaluators are selected from the academia having almost the same educational background and experience. For those who have not passed a related course, we had a personal teaching session. Also, they had a general teaching session to adjust their level of knowledge.

Considering the problem domains, the case studies should be almost in the same size and complexity level. Otherwise, a risk of validity for the results arises. Ideally, the same problem can be used for two groups, which is adopted in this research.

Regarding the case studies, the specific selection for case studies could have influence on the obtained results. So, some of them can be more suitable to handle with DSML which also constitute a threat to validity. To reduce this risk, several actions take place. First, several case studies are considered from various problem domains to cover different features and capabilities. Second, the same case studies are used for the development of all team members and the average times for different phases of their development are calculated to adjust the risk. Third, both of the groups use the same multi-case study which reduces the effect of this threat to validity. Finally, we have a briefing for the case studies in the evaluation framework which adjusts the level of knowledge of the developers from the problem domain.

The representation of the problem for the evaluation is also important and can have influence on the result. The problem can be defined using a questionnaire, modifying or changing the available code, interpretation of available code, and/or writing program from scratch. In our study, among those approaches, we have used a questionnaire, modeling and writing programs for case studies (for group A and B, respectively), and extending the programs with a new feature (as a change) to cover different aspects of representation for the cases. Moreover, the order of the questions can also affect the result, because the same user learns the issues in the early questions and can help him/her in solving the later questions (another construction validity). To reduce this effect as much as possible, we have used the same order of the cases in each group and for each developer.

In addition, selected tools which are used by developers can affect the results which cause a potential risk for the validity. For developers in group A, only tool is SEA_ML, so, there is no threat to the validity for their results. However, group B can use various tools and methodologies for their MAS development. Although their tools for analysis and design can be different (such as UML and MAS methodologies, e.g., Prometheus, Gaia, and Tropos), developers in group B utilize the same APIs and tools for the target agent platforms (JADEX and JACK). Based on the results, it can be said that the divergence appeared on group B developers' tool selections for analysis and design has minor effect on their MAS development process and final throughput comparing between two groups, so, it has less risk to the validity of the result.

Finally, to reduce the risk of validity for the evaluation and to make the results less dependent on the domain of the case studies, the results are collected for each case study and the average number is calculated for all of them.

8 Related work

The evaluation of a software system is one of the challenging issues in the field of software engineering. Although there are some textbooks (Abran 2010) and papers (Basili et al. 1999; Carver et al. 2010; Bayona-Oré et al. 2014) for evaluation of the software systems in general, the studies which evaluate DSMLs are rare (Marin et al. 2010). Taking into account general DSL studies with including an evaluation, for instance, in Wile (2004), the author describes experiences and lessons learned (successes and failures) from employing three real DSLs during 10 years. These DSLs are "Military Message Experiment (MME)," "Replenishment at Sea (RAS)," and "Survey Instrument Creator (SIC)." However, only the last one is evaluated formally. In the evaluation of this system, five key objectives are considered that the project purposed to achieve, e.g., help the user to design the instrument, using users' knowledge, and so on. At the end of the evaluation, the features which the evaluators liked and did not like are listed, e.g., graphical tool, ease to design the system, and so on. In general, the lessons learned from developing these DSLs are divided into three groups: technical issues, organizational issues, and social issues. In this way the author conveys his experiences to different groups of people who deal with DSLs. So, the author suggests a procedure to consider the five innovations that the application engineers are being asked to learn and use in the proposed methodology: artifacts, language, tools, infrastructure and methodology.

Inspiring mainly from the study in Wile (2004), the authors of Kahlaoui et al. (2008) propose some success factors for DSMLs and their assessment criteria. The success factors are considered for DSML developers, and assessment criteria (for quality) are considered for DSML users. They provide a list of success factors and a technique to convert them into assessment criteria. The success factors are considered from using three different perspectives: Quality of Models, Quality of Modeling Languages (Bobkowska 2005; Paige et al. 2000), and DSML design experiences (Wile 2004). Twelve success factors have been identified by combining the results of work carried out in the three dimensions described previously. The authors believe that DSMLs have four main elements: Abstract syntax (concepts and relations), Concrete syntax (notation; text/graphics), Semantics (gives a meaning to the concepts and the relations), and Views (perspectives of software). They also believe that the proposed success factors have effect on some of the mentioned elements (Kahlaoui et al. 2008). Considering these effects they offer some criteria to assess and select DSML (for end users). Work given in both Wile (2004) and Kahlaoui et al. (2008) guide our work in determination of some of the assessment criteria and use of some DSML success factors as previously discussed in Sects. 3 and 6, respectively.

Another study in literature is Tekinerdogan et al. (2011) in which the authors apply MDA techniques on process sensitive embedded user assistance with aim to provide the end user the necessary guidance based on the state of the process that is being followed. In their study two case applications are examined; Message Management System (MMS) and Listing and Listening of Voice Records (LLVR). To evaluate the approach, the authors examine the cost-effectiveness of the system. To this end, the tool has been provided to a software development group to define "user assistance" for LLVR. The software development group had a standard background in Computer Science and Software Engineering. The time for using the necessary activities of the tool has been recorded for analysis. Based on a general "cost model" for developing embedded user assistance they have provided an evaluation of the tool. The cost model of our evaluation, is in fact an extended version of the model given in this study. We used the extended model during execution with data collection.

In the study of Kosar et al. (2010), a GPL and a DSL are compared with an empirical study to measure programmer's understanding of DSL and GPL programs on the same problem domain. The domain is graphical user interface production. The GPL is C# and the DSL is XAML. The comparison is conducted via an experiment with two groups of programmers who have answered a questionnaire, one group for GPL and another for DSL considering the same problem domain. The answer sheets of evaluators were processed, and the uncompleted results were eliminated from the rest of analysis. Then the success rates were calculated for each question of GPL and DSL (applying mean and standard deviation). Also, extending this study with some other empirical researches, family of experiments is presented in Kosar et al. (2012) by comparing results of using DSLs and related GPL application libraries. Team preparation and execution in our work have similarities with these studies considering preparation of the teams, and questionnaire building. For example, both studies use case studies and have two groups of evaluators. However, structure of the case study presentation and formalization of questionnaire differ in our study according to the MAS developer profiles and MAS DSML viewpoint specializations.

Barisic et al. (2011a) outlines why usability, i.e., Quality in Use, is the most relevant to be evaluated for the DSL under development. It also gives an appropriate definition of DSL in order to be able to apply evaluation practices from Human–Computer interaction community and explains why existing practices for evaluating GPLs are not sufficient in this case. A case study on how to perform usability evaluation of DSL with real users by taking into consideration expert users, as well as novices is given in Barisic et al. (2011b). While studies in Barisic et al. (2011a, b) have similarities with our study in the way of handling the quality of DSL usability, the evaluation in our work specifically concentrates on the usability quality of MAS DSMLs instead of a general perspective.

In Barisic et al. (2012b) the authors present how to apply a set of patterns to perform DSL evaluation. These patterns outline different roles relevant for the development and evaluation of DSLs. All the patterns are followed by ongoing DSL example to give an idea of their application to real DSL cases. Barisic (2013) proposes a process of the development of DSL that includes a new role: Usability Engineer that is responsible to prepare and evaluate produced a solution in an agile way, by capturing language user's preferences. Following this approach, it is expected to identify a usability failure early in the development cycle and get the means to prepare a final evaluation that will give us meaningful and generalizable results. In order to experimentally evaluate a DSL, there is a need to know what are the criteria involved, understand the notion of quality from the relevant perspectives and understand the experimental process itself. This complex challenge is covered in Barisic et al. (2012a) by the general model for DSL experimental evaluation that served as a set of proof of concept instantiations. Patterns in these studies cover agile development process, iterative user-centered design and experimental evaluation design. Although the organization of Barisic et al.'s experimental evaluation resembles the multi-case study approach brought in our study, quality dimension of our framework supports more criteria especially in the general DSML assessment comparing with the patterns covered in Barisic et al. (2012b).

Finally, there is another study (Kahraman and Bilgen 2013) which presents a Framework for Qualitative Assessment of DSLs (called FQAD). FQAD is used for determining the perspective of the evaluator, understanding the goal of the assessment and selecting the fundamental DSL quality characteristics to guide the evaluator in the process. This framework adapts and integrates the ISO/IEC 25010:2011 standard (ISO/IEC 2011), CMMI maturity level evaluation approach (CMU/SEI 2010) and the scaling approach used in DESMET (Kitchenham et al. 1997) into a perspective-based assessment. However, the authors of this study focus on the evaluation of a DSL by solely considering the quality of a DSL, and hence FQAD does not include any performance measurement and/or tool assessment which differ from our approach.

During the discussion on the state of the art of MAS DSMLs in Sect. 2, it has been clearly revealed that the work reported in this paper presents the first comprehensive MAS DSML evaluation framework within the agent-oriented software engineering (AOSE) research field, which also covers the usage of the framework in the real evaluation of one of the well-known MAS DSMLs with including both qualitative and quantitative assessment of the applied methodology and the generated artifacts. In addition, we believe that our evaluation framework also contributes to the above-mentioned general-purpose DS(M)L evaluation studies especially with introducing a new assessment approach and a set of criteria which can be served for the evaluation of DSMLs by taking into account agent system viewpoints both in static and dynamic dimensions.

The evaluation framework introduced in this paper benefited from the outcomes of many of the previous DS(M)L evaluation efforts (e.g., Wile 2004; Kahlaoui et al. 2008; Tekinerdogan et al. 2011; Kosar et al. 2012; Kahraman and Bilgen 2013) during the derivation of criteria and the application of the framework (as indicated in the various sections of this paper). However, just straightforward application of those approaches or assessment based on some sort of synthesis of their features without considering the MAS specifications would expose drawbacks and inefficiency due to the unordinary, distributive and complex nature of systems based on software agents. Instead, our work shows that directly supporting the various aspects of MAS modeling during DSML evaluation such as agent behaviors, composite agent interactions and resource allocation inside rapidly changing agent environments facilitates the selection and utilization of MAS DSMLs and hence the assessment of the DSML usage in the development of software systems that are composed of both autonomous and proactive entities with social ability. Within this context, reported work differentiates from those existing general DS(M)L evaluation efforts and contributes to their field by specializing on the MAS domain. Moreover, as can be seen from the structure of the above given studies, they mostly (e.g., Kahlaoui et al. 2008; Barisic et al. 2011b; Kahraman and Bilgen 2013) focus on the evaluation of a DSL just considering the quality. Our work does not only take the quality assessment into the consideration, but also covers the tool assessment regarding its complexity level and development performance along with the evaluation of agent artifacts. We think that presenting a set of quantitative measurements in addition to the experience-based assessment facilitates the MAS DSML evaluation and helps agent developers on the use of MAS DSMLs during an MDD.

9 Conclusion

In this paper we have provided a systematic approach for evaluating MAS DSMLs. The approach is based on an evaluation framework that includes different dimensions and criteria for supporting the evaluation. We have adopted SEA_ML, a representative MAS

DSML to illustrate the evaluation framework and the corresponding approach. Using a multi-case study approach, we have selected four different case studies in which MASs are developed. In our multi-case study approach, one group of developers adopted the SEA_ML to design the cases while the other group uses general-purpose modeling languages. During the study, the values for the criteria in the evaluation framework were collected. For supporting the qualitative analysis, we have prepared questionnaires and the feedback of both groups with respect to the evaluation framework criteria were collected. We believe that the conducted multi-case study research with the four case studies illustrated that the evaluation framework and the corresponding systematic approach can be used in evaluating a MAS DSML both from the quantitative and qualitative dimensions. The results from both dimensions justified the adoption of the systematic evaluation approach to evaluate MAS DSMLs. At this point, one can argue that a DSML is already expected to enhance the development. MAS software development adopting a DSML will have benefits compared to manual development and/or using a GPL. So, it can be thought that some parts of the evaluation performed on this study provide the justification of using an MAS DSML instead of a straightforward and GPL-based development. However, apart from this justification, the framework and the multi-case study approach presented in this work can also be employed in comparing MAS DSMLs. For instance, the use of another MAS DSML, say DSML4MAS (Hahn 2008), can be evaluated with the same multi-case study and hence the achieved results can be compared with the ones given in Sect. 6 of this paper for SEA_ML usage. That may assist the determination of the MAS DSML which is most appropriate for a specific MAS development. Such usage is probably one of the most important benefits of our evaluation framework brought to the AOSE community.

It is worth indicating that, although the evaluation results show that SEA_ML is effective, developing a DSML is a difficult and time-consuming task. It needs a high level of expertise to develop a good language and tool. So, tradeoffs should be considered. Also, its learning curve for users should be taken into the consideration. The expense of developing a DSML are high and its breaking point for benefits can be far. In other words, the time and money spent in developing a DSML should return in a logical time period. These efforts can be the development and learning procedure of the DSML. These efforts are retaliated by the benefits of saving the time and effort to develop in several future software developments. But the key question is how much time later or how many projects later this tradeoff will be in our favor. This question should be answered and the answer should satisfy one to start developing a DSML.

Although SEA_ML was considered quite effective, one important result of the study was also the identification of the points that need improvement in the language. For SEA_ML this was described as a need for transforming the changes in the generated code back to the models. In fact, this is our future work to extend the language by means of round-trip engineering. We believe that a systematic evaluation approach that we have presented here can also characterize and evaluate existing or newly defined DSMLs with respect to their effectiveness for developing MASs. In our future work we will enhance SEA_ML with respect to the results of the evaluation. Further we aim at evaluating also other MAS DSMLs and where necessary enhancing the framework. Finally, the current assessment of the quantitative properties such as the number of key elements supported by the abstract syntax of a MAS DSML mostly helps the language analysis. Utilization of such quantitative properties during detailed evaluation of MAS DSML usage is also future work.

Acknowledgments This study is partially funded by the Scientific and Technological Research Council of Turkey (TUBITAK) Electric, Electronic and Informatics Research Group (EEEAG) under Grant 109E125. The authors wish to thank the former members of the SEA_ML project, Dr. Marjan Mernik, Dr. Tomaz Kosar, Sinem Getir and Sebla Demirkol for their valuable contributions on design and implementation of SEA_ML. Finally, the authors would like to thank the participants, various Ph.D. and M.Sc. students, who actively got involved in the language evaluation and helped to the realization of the empirical study presented in this paper.

References

Abran, A. (2010). *Software metrics and software metrology*. New Jersey: Wiley and IEEE-CS Press.

AOS. (2001). Agent Oriented Software Pty., Ltd. JACK Environment. http://www.aosgrp.com/products/jack/. Last Accessed August 2015.

Barisic, A. (2013). Evaluating the quality in use of domain-specific languages in an agile way. In *Proceedings of the doctoral symposium at 16th international conference on model driven engineering languages and systems (MODELS)*, September, 2013, CEUR-WS.

Barisic, A., Amaral, V., Goulao, M., & Barroca, B. (2011a). How to reach a usable DSL? Moving toward a systematic evaluation. *Electronic Communications of the EASST, 50*, 1–12.

Barisic, A., Amaral, V., Goulao, M., & Barroca, B. (2011b). Quality in use of domain-specific languages: A case study. In *Proceedings of the 3rd ACM SIGPLAN workshop on evaluation and usability of programming languages and tools* (pp. 65–72). PLATEAU'11, New York, USA.

Barisic, A., Amaral, V., Goulao, M., & Barroca, B. (2012a). Evaluating the usability of domain-specific languages. In M. Mernik (Ed.), *Formal and practical aspects of domain-specific languages: Recent developments* (pp. 386–407). Hershey: IGI Global.

Barisic, A., Amaral, V., Goulao, M., & Monteiro, M. P. (2012b). Patterns for evaluating usability of domain-specific languages. In *Proceedings of the 19th conference on pattern languages of programs (PLoP)*, SPLASH.

Basili, V., Shull, F., & Lanubile, F. (1999). Building knowledge through families of experiments. *IEEE Transactions on Software Engineering, 25*(4), 456–473.

Bayona-Oré, S., Calvo-Manzano, J. A., Cuevas, G., & San-Feliu, T. (2014). Critical success factors taxonomy for software process deployment. *Software Quality Journal, 22*(1), 21–48.

Bellifemine, F., Poggi, A., & Rimassa, G. (2001). Developing multi-agent systems with a FIPA-compliant agent framework. *Software: Practice and Experience, 31*(2), 103–128.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American, 284*(5), 34–43.

Bernon, C., Cossentino, M., Gleizes, M.-P., Turci, P., & Zambonelli, F. (2005). A study of some multi-agent meta-models. *Lecture Notes in Computer Science, 3382*, 62–77.

Beydoun, G., Low, G. C., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J. J., Pavon, J., & Gonzalez-Perez, C. (2009). FAML: A generic metamodel for MAS development. *IEEE Transactions on Software Engineering, 35*(6), 841–863.

Bobkowska, A. (2005). A methodology of visual modeling language evaluation. In *SOFSEM 2005: Theory and practice of computer systems*. *Lecture Notes in Computer Science* (Vol. 3381).

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., & Mylopoulos, J. (2004). Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems, 8*(3), 203–236.

Carver, J., Jaccheri, L., Morasca, S., & Shull, F. (2010). A checklist for integrating student empirical studies with research and teaching goals. *Empirical Software Engineering, 15*(1), 35–59.

Challenger, M., Demirkol, S., Getir, S., Mernik, M., Kardas, G., & Kosar, T. (2014). On the use of a domain-specific modeling language in the development of multi-agent systems. *Engineering Applications of Artificial Intelligence, 28*, 111–141.

Challenger, M., Getir, S., Demirkol, S., & Kardas, G. (2011). A domain specific metamodel for semantic web enabled multi-agent systems. *Lecture Notes in Business Information Processing, 83*, 177–186.

Ciobanu, G., & Juravle, C. (2012). Flexible software architecture and language for mobile agents. *Concurrency and Computation: Practice and Experience, 24*(6), 559–571.

Clark, T., Evans, A., Sammut, P., & Willans, J. (2004). Language driven development and MDA. *MDA Journal*, 2–13. http://www.bptrends.com/bpt/wp-content/publicationfiles/10-04%20COL%20MDA%20-%20Frankel%20-%20Xactium.pdf. Last Accessed August 2015.

CMU/SEI. (2010). Software Engineering Institute, CMMI for Acquisition, Version 1.3, Technical Report, CMU/SEI-2010-TR-032, Carnegie Mellon (November 2012).

CMU/SEI. (2010). Software Engineering Institute, CMMI for Acquisition, Version 1.3, Technical Report, CMU/SEI-2010-TR-032, Carnegie Mellon (November 2012).

Demirkol, S., Challenger, M., Getir, S., Kosar, T., Kardas, G., & Mernik, M. (2013). A DSL for the development of software agents working within a semantic web environment. *Computer Science and Information Systems, Special Issue on Advances in Model Driven Engineering, Languages and Agents, 10*(4), 1525–1556.

Demirkol, S., Getir, S., Challenger, M., & Kardas, G. (2011). Development of an agent based E-barter System. In: *International symposium on innovations in intelligent systems and applications (INISTA)* (pp. 193–198). IEEE Computer Society, Istanbul-Turkey.

Demirli, E., & Tekinerdogan, B. (2011). Software language engineering of architectural viewpoints. In *Proceeding of the 5th European Conference on Software Architecture (ECSA 2011)*, *Lecture Notes on Computer Science* (Vol. 6903, pp. 336–343).

Eclipse. (2004). The Eclipse Foundation, Eclipse Integrated Development Environment (IDE). http://www.eclipse.org/. Last Accessed August 2015.

Fister, I, Jr, Fister, I., Mernik, M., & Brest, J. (2011). Design and implementation of domain-specific language easytime. *Computer Languages, Systems and Structures, 37*(4), 151–167.

Fowler, M. (2011). *Domain-specific languages*. Boston: Addison-Wesley.

Fuentes-Fernandez, R., Garcia-Magarino, I., Gomez-Rodriguez, A. M., & Gonzalez-Moreno, J. C. (2010). A technique for defining agent-oriented engineering processes with tool support. *Engineering Applications of Artificial Intelligence, 23*(3), 432–444.

Garcia, A. F., Sant'Anna, C., Figueiredo, E., Kulesza, U., Pereira de Lucena, C. J., & von Staa, A. (2005). Modularizing design patterns with aspects: A quantitative study. In *Proceedings of the 4th international conference on Aspect-oriented software development (AOSD 2005)* (pp. 3–14).

Gascuena, J. M., Navarro, E., & Fernandez-Caballero, A. (2012). Model-driven engineering techniques for the development of multi-agent systems. *Engineering Applications of Artificial Intelligence, 25*(1), 159–173.

Getir, S., Challenger, M., Demirkol, S., & Kardas, G. (2012). The semantics of the interaction between agents and web services on the semantic web. In *Proceedings of the 7th IEEE international workshop on engineering semantic agent systems (ESAS 2012), held in conjunction with the 36th IEEE signature conference on computers, software, and applications (COMPSAC 2012)* (pp. 619–624).

Getir, S., Challenger, M., & Kardas, G. (2014). The formal semantics of a domain-specific modeling language for semantic web enabled multi-agent systems. *International Journal of Cooperative Information Systems, 23*(3), 1450005.

Gray, J., Tolvanen, J.-P., Kelly, S., Gokhale, A., Neema, S., & Sprinkle, J. (2007). Domain-specific modeling. In P. A. Fishwick (Ed.), *Handbook of dynamic system modeling* (pp. 1–7). Boca Raton: CRC Press.

Hahn, C. (2008). A domain specific language for multiagent systems. In *Proceedings of the 7th autonomous agents and multiagent systems conference (AAMAS'08), Estoril, Portugal* (pp. 233–240).

Hahn, C., Madrigal-Mora, C., & Fischer, K. (2009). A platform-independent metamodel for multiagent systems. *Autonomous Agents and Multi-Agent Systems, 18*(2), 239–266.

Hahn, C., Nesbigall, S., Warwas, S., Zinnikus, I., Fischer, K., & Klusch, M. (2008). Integration of multiagent systems and semantic web services on a platform independent level. In *Proceedings of IEEE/WIC/ACM international conference on web intelligence and intelligent agent technology (WI-IAT 2008), Sydney, Australia* (pp. 200–206).

Hannemann, J., & Kiczales, G. (2002). Design patterns implementation in Java and AspectJ. In *Proceeding of object oriented programming systems languages and applications (OOPSLA'02)* (pp. 161–173).

ISO/IEC. (1998). International Organization for Standardization (Ed.), ISO/IEC TR 15504-2. Information Technology-Software Process Assessment—Part 2: A reference model for processes and process capability. Case Postale 56, CH-1211 Geneva, Switzerland.

ISO/IEC. (2011). ISO/IEC 25010. Systems and software engineering systems and software quality requirements and evaluation (SQuaRE), System and software quality models, International Standards Organization/International Electrotechnical Commission.

Jackson, D. (2002). Alloy: A lightweight object modeling notation. *ACM Transactions on Software Engineering and Methodology, 11*(2), 256–290.

Jouault, F., Allilaire, F., Bezivin, J., & Kurtev, I. (2008). ATL: A model transformation tool. *Science of Computer Programming, 72*(1–2), 31–39.

Kahlaoui, A., Abran, A., & Lefebvre, E. (2008). DSML success factors and their assessment criteria. *Metrics News, 13*(1), 43–51.

Kahraman, G., & Bilgen, S. (2013). A framework for qualitative assessment of domain-specific languages. *Software & Systems Modeling*. doi:10.1007/s10270-013-0387-8.

Kardas, G., Challenger, M., Yildirim, S., & Yamuc, A. (2012). Design and implementation of a multi-agent stock trading system. *Software Practice and Experience, 42*, 1247–1273.

Kardas, G., Goknil, A., Dikenelli, O., & Topaloglu, N. Y. (2009). Model driven development of semantic web enabled multi-agent systems. *International Journal of Cooperative Information Systems, 18*(2), 261–308.

Kelly, S., & Tolvanen, J.-P. (2008). *Domain-specific modeling- enabling full code generation*. Los Alamitos: IEEE Computer Society Publications.

Khedker, U. P. (1997). *What makes a good programming language?* Technical Report TR-97-upk-1, Department of Computer Science, University of Pune.

Kirstan, S., & Zimmermann, J. (2010). Evaluating costs and benefits of model-based development of embedded software systems in the car industry—Results of a qualitative Case Study. In *Proceedings of ECMFA 2010 workshop C2 M:EEMDD-from Code Centric to Model Centric: Evaluating the effectiveness of MDD* (pp 18–29).

Kitchenham, B. A., Linkman, S., & Law, D. (1997). DESMET: A methodology for evaluating software engineering methods and tools. *Computing and Control Engineering Journal, 8*(3), 120–126.

Kosar, T., Mernik, M., & Carver, J. C. (2012). Program comprehension of domain-specific and general-purpose languages: Comparison using a family of experiments. *Empircal Software Engineering, 17*, 276–304.

Kosar, T., Oliveira, N., Mernik, M., Varanda Pereira, M. J., Crepinsek, M., Carneiro da Cruz, D., & Henriques, P. R. (2010). Comparing general-purpose and domain-specific languages: An empirical study. *Computer Science and Information Systems, 7*(2), 247–264.

Marin, B., Pastor, O., & Abran, A. (2010). Towards an accurate functional size measurement procedure for conceptual models in an MDA environment. *Journal of Data and Knowledge Engineering, 69*(5), 472–490.

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., & Sycara, K. (2004). *OWL-S: Semantic markup for web services*. W3C Member Submission. http://www.w3.org/Submission/OWL-S/. Last Accessed August 2015.

Mernik, M., Heering, J., & Sloane, A. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys, 37*(4), 316–344.

Mernik, M., & Zumer, V. (2005). Incremental programming language development. *Computer Languages, Systems and Structures, 31*(1), 1–16.

MOFScript. (2005). The Eclipse Foundation, MOFScript model-to-text transformation language and tool. http://www.eclipse.org/gmt/mofscript/. Last Accessed August 2015.

Mohagheghi, P., & Dehlen, V. (2008). Where is the proof? A review of experiences from applying MDE in industry. In: I. Schieferdecker, & A. Hartman (Eds.) ECMDA-FA 2008. *Lecture notes in computer science* (Vol. 5095, pp. 432–443).

Mohagheghi, P., Gilani, W., Stefanescu, A., & Fernandez, M. A. (2013). An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases. *Empirical Software Engineering, 18*, 89–116.

Monteiro, M. P., & Fernandes, J. M. (2005). Towards a catalog of aspect oriented refactorings. In *Proceedings of the 4th international conference on aspect oriented software development (AOSD'05)* (pp. 111–122).

OCL. (2012). Object Management Group, Object Constraint Language (OCL). http://www.omg.org/spec/OCL/2.3.1/. Last Accessed August 2015.

Odell, J., Nodine, M., & Levy, R. (2005). A metamodel for agents, roles, and groups. Agent-oriented software engineering. *Lecture Notes in Computer Science, 3382*, 78–92.

Oldevik, J., Neple, T., Gronmo, R., Aagedal, J., & Berre, A. J. (2005). Toward standardized model to text transformations. *Lecture Notes in Computer Science, 3748*, 239–253.

Omicini, A., Ricci, A., & Viroli, M. (2008). Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems, 17*(3), 432–456.

Padgham, L., & Winikoff, M. (2004). *Developing intelligent agent systems: A practical guide*. New York: Wiley.

Paige, R. F., Ostroff, J. S., & Brooke, P. J. (2000). Principles for modeling language design. *Information and Software Technology, 42*(10), 665–675.

Pavon, J., Gomez-Sanz, J. J., & Fuentes, R. (2006). Model driven development of multi-agent systems. *Lecture Notes in Computer Science, 4066*, 284–298.

Pešović, D., Vidaković, M., Ivanović, M., Budimac, Z., & Vidaković, J. (2011). Usage of agents in document management. *Computer Science and Information Systems, 8*, 193–210.

Pokahr, A., Braubach, L., & Lamersdorf, W. (2005). JADEX: A BDI reasoning engine. In R. H. Bordini, et al. (Eds.), *Multi-agent programming languages, platforms and applications* (pp. 149–174). Berlin: Springer.

Rao, A., & Georgeff, M. (1995). BDI agents: From theory to practice. In *Proceedings of the 1st international conference on multi-agent systems (ICMAS-95), San Francisco* (pp. 312–319).

Rougemaille, S., Migeon, F., Maurel, C., & Gleizes, M.-P. (2007). Model driven engineering for designing adaptive multi-agent systems. *Lecture Notes in Artificial Intelligence, 4995*, 318–332.

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering, 14*, 131–164.

Russell, S. J., & Norvig, P. (2003). *Artificial intelligence: A modern approach* (2nd ed.). USA: Pearson Education.

Saritas, H. B., & Kardas, G. (2014). A model driven architecture for the development of smart card software. *Computer Languages, Systems and Structures, 40*(2), 53–72.

Schmid, K., & John, I. (2002). Developing, validating and evolving an approach to product line benefit and risk assessment. In *Proceedings of the 28th Euro-micro conference (EUROMICRO'02)* (pp. 272–283).

Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. *IEEE Computer, 39*(2), 25–31.

Shadbolt, N., Hall, W., & Berners-Lee, T. (2006). The semantic web revisited. *IEEE Intelligent Systems, 21*(3), 96–101.

Shih, D. H., Huang, S. Y., & Yen, D. C. (2005). A new reverse auction agent system for M-commerce using mobile agents. *Computer Standards and Interfaces, 27*, 383–395.

Smith, R. (1980). The Contract Net protocol: High Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers, 29*(12), 1104–1113.

Sprinkle, J., Mernik, M., Tolvanen, J.-P., & Spinellis, D. (2009). Guest editors' introduction: What kinds of nails need a domain-specific hammer? *IEEE Software, 26*(4), 15–18.

Sycara, K. (1998). Multiagent systems. *AI Magazine, 19*(4), 79–92.

Sycara, K., Paolucci, M., Ankolekar, A., & Srinivasan, N. (2003). Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics, 1*(1), 27–46.

Tekinerdogan, B., Bozbey, S., Mester, Y., Turançiftci, E., & Alkışlar, L. (2011). An aspect-oriented tool framework for developing process-sensitive embedded user assistance systems. In *Transactions on aspect-oriented software development VIII. Lecture notes computer science* (Vol. 6580, pp. 196–220).

Tekinerdogan, B., & Demirli, E. (2013). Evaluation framework for software architecture viewpoint languages. In *Proceedings of 9th international ACM Sigsoft conference on the quality of software architectures (QoSA 2013)* (pp. 89–98).

van Deursen, A., Klint, P., & Visser, J. (2000). Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices, 35*(6), 26–36.

Varanda-Pereira, M. J., Mernik, M., Da-Cruz, D., & Henriques, P. R. (2008). Program comprehension for domain-specific languages. *Computer Science and Information Systems, 5*(2), 1–17.

Warmer, J., & Kleppe, A. G. (1998). *The object constraint language: Precise modeling with UML.* Boston: Addison-Wesley.

Wile, D. (2004). Lessons learned from real DSL experiments. *Science of Computer Programming, 51*, 265–290.

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review, 10*(2), 115–152.

Zambonelli, F., Jennings, N. R., & Wooldrige, M. (2003). Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodologies, 12*(3), 317–370.

Moharram Challenger received his B.Sc., and M.Sc. degrees in Computer Engineering and Software Engineering from IAU-Shabestar and IAU-Arak Universities (Iran) in 2001 and 2005, respectively. Since 2006, he has been a tenure-track faculty member, as a lecturer, at computer engineering department, IAU-Shabestar University. He is currently a Ph.D. candidate at Ege University, International Computer Institute. He is also R&D director of UNIT IT Ltd. Company in which they are leading an international ITEA Project, called ModelWriter, among Belgium, Turkey and France. His research interests include domain-specific (modeling) languages, multi-agent and distributed systems with a current focus on the semantics of DSMLs. Moharram is also a student member of the IEEE and ACM.

Geylani Kardas received his B.Sc. in Computer Engineering and both M.Sc., and Ph.D. degrees in Information Technologies from Ege University in 2001, 2003 and 2008, respectively. He is currently an associate professor at Ege University, International Computer Institute. His research interests include agent-oriented software engineering, model-driven software development, and domain-specific (modeling) languages. He has authored or co-authored over 50 published papers in these areas. In the recent past, he has also worked as the principle investigator, researcher or consultant in various national and international R&D projects funded by both governments and private agencies. He is a member of the ACM.

Bedir Tekinerdogan received his M.Sc. degree (1994) and a Ph.D. degree (2000) in Computer Science, both from the University of Twente, The Netherlands. From 2003 until 2008 he was a faculty member at University of Twente, after which he joined Bilkent University until 2015. At Bilkent he has founded and led the Bilkent Software Engineering Group which aimed to foster research and education on software engineering in Turkey. Currently he is full professor and chair of the Information Technology group at Wageningen University, The Netherlands. He has more than 20 years of experience in software engineering research and education. He has been active in many different research and consultancy projects with various software engineering companies and is the author of around 200 scientific publications on different topics in software engineering. His key research topics are software architecture design, model-driven software development, software product line engineering, and aspect-oriented software development.